DIALOG(R)File 351:Derwent WPI

014319621 **Image available**
WPI Acc No: 2002-140323/200219
XRPX Acc No: N02-106173

Instruction generation method for directed adjacency graph for computer graphics, involves generating leaf instructions for those graphic objects that are active at one or more pixel locations

Patent Assignee: CANON KK (CANO); MOORE K J (MOOR-I)
Inventor: MOORE K J
Number of Countries: 003 Number of Patents: 004
Patent Family:

Patent No	Kind	Date	Applicat No	Kind	Date	Week	
AU 200135163	A	20011025	AU 200135163	A	20010412	200219	B
JP 2002056396	A	20020220	JP 2001120228	A	20010418	200219	
US 20020015039	A1	20020207	US 2001836226	A	20010418	200219	
AU 760826	B	20030522	AU 200135163	A	20010412	200338	

Priority Applications (No Type Date): AU 20006971 A 20000418
Patent Details:

Patent No	Kind	Lan	Pg	Main IPC	Filing Notes
AU 200135163	A		97	G06T-009/40	
JP 2002056396	A		134	G06T-011/40	
US 20020015039	A1			G06T-015/40	
AU 760826	B			G06T-009/40	Previous Publ. patent AU 200135163

Abstract (Basic): AU 200135163 A

NOVELTY - A group of one or more pixel locations are determined and a portion of the graph that passes-up the data is determined for each group, in accordance with activities of parent nodes. Instructions are generated for those nodes of the determined portion of the graph having active branches, and leaf instructions are generated for those graphic objects which are active at group of one or more pixel locations.

DETAILED DESCRIPTION - INDEPENDENT CLAIMS are also included for the following:

(a) Directed adjacency graph rendering method;

(b) Apparatus for generating instructions for a directed adjacency graph;

(c) Apparatus for rendering directed adjacency graph into a raster pixel image;

(d) Computer readable medium storing program for generating instructions for a directed adjacency graph;

(e) Computer readable medium storing computer program for rendering directed adjacency graph;

(f) Directed adjacency graph rendering program

USE - For generating instructions for directed adjacency graph such as expression tree, for rendering graphic object based images in object-based graphic system e.g. computer graphics and printing industries.

ADVANTAGE - Provides flexibility to cope up with tree-based graphical expression. Eliminates complex arrangement of clipping objects and thus reduces processing time.

DESCRIPTION OF DRAWING(S) - The figure shows a schematic block diagram representation of a computer system.

pp; 97 DwgNo 1/30

Title Terms: INSTRUCTION; GENERATE; METHOD; DIRECT; ADJACENT; GRAPH; COMPUTER; GRAPHIC; GENERATE; LEAF; INSTRUCTION; GRAPHIC; OBJECT; ACTIVE; ONE; MORE; PIXEL; LOCATE
Derwent Class: T01

International Patent Class (Main): G06T-009/40; G06T-011/40; G06T-015/40
International Patent Class (Additional): G06T-009/20; G09G-005/00;
G09G-005/36
File Segment: EPI
?

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-056396
(43)Date of publication of application : 20.02.2002

(51)Int.Cl. G06T 11/40
G09G 5/36

(21)Application number : 2001-120228
(22)Date of filing : 18.04.2001

(71)Applicant : CANON INC
(72)Inventor : MOORE KEVIN JOHN

(30)Priority
Priority number : 2000 PQ6971 Priority date : 18.04.2000 Priority country : AU

(54) PLOTTING METHOD AND DEVICE FOR GRAPHIC OBJECT IMAGE

(57)Abstract:
PROBLEM TO BE SOLVED: To provide a method for generating a command to a directed adjacent graph such as expression tree or the like for plotting to a raster pixel image having a plurality of scanning lines and a plurality of pixel positions in each scanning line.
SOLUTION: The expression tree comprises at least one master node and at least leaf node. Each of the master nodes expresses an operation and has a branch to the respective lower node. The leaf node expresses a graphic object. This device comprises a module 504 for determining the part of the expression tree for delivering a data to the expression tree according to the activity of a computing element and a module 506 for generating a command to the determined part of the expression tree, and a computing element command to the computing element of the part having the active branch of the expression tree and a leaf command for the active graphic object are generated therein.

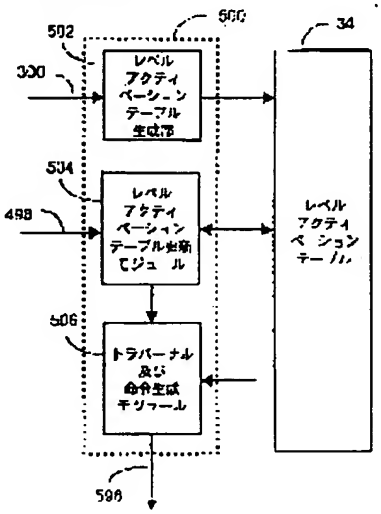


Fig. 5

LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2002－56396
（Ｐ2002－56396Ａ）
(43)公開日　平成14年２月20日(2002.2.20)

(51)Int.Cl.[7]	識別記号	Ｆ Ｉ	テーマコード(参考)
Ｇ０６Ｔ　11/40	２００	Ｃ０６Ｔ　11/40	２００Ａ　５Ｂ０８０
Ｇ０９Ｇ　5/36		Ｃ０９Ｇ　5/36	５２０Ｐ　５Ｃ０８２
			５３０Ｘ

審査請求　未請求　請求項の数33　ＯＬ　外国語出願　（全134頁）

(21)出願番号　特願2001－120228(Ｐ2001－120228)
(22)出願日　平成13年４月18日(2001.4.18)
(31)優先権主張番号　ＰＱ６９７１
(32)優先日　平成12年４月18日(2000.4.18)
(33)優先権主張国　オーストラリア（ＡＵ）

(71)出願人　000001007
キヤノン株式会社
東京都大田区下丸子３丁目30番２号
(72)発明者　ケビン　ジョン　ムーア
オーストラリア国　2113　ニュー　サウス　ウェールズ州，　ノース　ライド，　トーマス　ホルト　ドライブ　１　キヤノン　インフォメーション　システムズ　リサーチ　オーストラリア　プロプライエタリー　リミテッド　内
(74)代理人　100076428
弁理士　大塚　康徳　（外３名）

Ｆターム(参考)　5B080 GA25
5C082 BA12 DA22 DA87 MM02

(54)【発明の名称】　グラフィックオブジェクトイメージ描画方法及び装置

(57)【要約】　（修正有）
【課題】複数の走査線と、各走査線における複数の画素位置とを有するラスタ画素イメージへ描画するための、表現ツリーなどの有向隣接図表に対する命令を生成する方法、を提示する。
【解決手段】表現ツリーは１つ以上の親ノードと１つ以上のリーフノードを含む。親ノードの各々は演算を表し、それぞれの下方ノードへのブランチを有する。リーフノードは、それぞれグラフィックオブジェクトを表す。装置は演算子のアクティビティに従って、表現ツリーにデータを渡す表現ツリーの部分を決定するためのモジュール５０４を、また、表現ツリーの決定された部分に対する命令を生成するモジュール５０６を備える。ここで、その表現ツリーのアクティブなブランチを有する部分の演算子に対して演算子命令が、また、アクティブなグラフィックオブジェクトのためのリーフ命令が生成される。

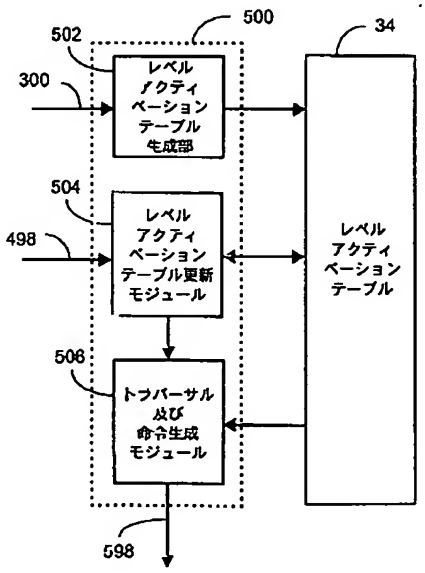


Fig. 5

【特許請求の範囲】

【請求項１】　有向隣接図表のための命令を生成する方法であって、前記有向隣接図表は１つ以上の親ノードと１つ以上のリーフノードを備え、それぞれの親ノードは演算子を表して、それぞれにぶらさがるブランチを有し、それぞれのリーフノードはグラフィックオブジェクトを表しており、前記方法が、

１つ以上の画素位置のグループを決定し、

それぞれの前記グループのために、演算子のアクティビティに従って前記有向隣接図表の一部を決定し、該有向隣接図表の該部分は、有向隣接図表へデータを渡す部分であり、

それぞれの前記グループのために、有向隣接図表の前記決定された部分への命令を生成し、ここで、演算子命令はアクティブなブランチを有する有向隣接図表の決定された部分のそれらの演算子のために生成され、リーフ命令は、１つ以上の画素位置の前記グループでアクティブなそれらのグラフィックオブジェクトのために生成されることを特徴とする方法。

【請求項２】　演算子のアクティビティと親ノードのブランチのアクティビティに関するデータを格納するテーブルを生成するステップを備えることを特徴とする請求項１に記載の方法。

【請求項３】　前記決定ステップは、それぞれの前記グループのために前記テーブルに格納された前記データを更新するサブステップを備えることを特徴とする請求項２に記載の方法。

【請求項４】　前記命令を生成するステップは、有向隣接図表の前記部分をなぞり、有向隣接図表の前記部分において、アクティブな演算子のための命令とリーフ値の命令とを生成するサブステップを含むことを特徴とする請求項１に記載の方法。

【請求項５】　前記有向隣接図表は表現ツリーであることを特徴とする請求項１に記載の方法。

【請求項６】　前記親ノードは２項演算子を表すことを特徴とする請求項１に記載の方法。

【請求項７】　表現ツリーを生成する方法であって、前記表現ツリーは、１つ以上のバイナリノードと複数のリーフノードを有する複数のノードを有し、ここで、それぞれの前記バイナリノードはそれにぶらさがる左ブランチと右ブランチとを有して該２つのぶらさがるノードへの２進演算を表し、それぞれの前記ノードは、１個以上の図形オブジェクトが重なった図形オブジェクトを表し、前記図形オブジェクトの重なりの各々が、左ノード領域、共通領域、および右ノード領域を含み、前記方法が、

１つ以上の画素位置のグループを決定し、

それぞれの前記グループについて、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断し、

それぞれの前記グループについて、前記表現ツリーをなぞり、ここで、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記左ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの左ブランチがアクティブで右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチにぶらさがるノードへなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記右ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチにぶらさがるノードへなぞられ、

それぞれの前記グループのために、前記アクティブな左右ブランチを有する前記なぞられたバイナリノードのための演算子命令、およびなぞられたリーフノードのためのリーフ値命令を生成することを特徴とする方法。

【請求項８】　前記なぞるステップは表現ツリーをなぞり、前記先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリノードの下方ノードの共通領域において、前記先になぞられたバイナリノードの前記左ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合、前記先になぞられたバイナリノードの左ブランチの下方ノードへとなぞられ、また、先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの前記左ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリノードの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すオブジェクトを隠さない場合、前記先になぞられたあらゆるバイナリノードの右ブランチの下方ノードへなぞられることを特徴とする請求項７に記載の方法。

【請求項９】　表現ツリーから、複数の走査線と各走査線に複数の画素位置を有するラスタ画素イメージを描画する方法であって、前記表現ツリーは複数のノードに１つ以上のバイナリノードと複数のリーフノードを有し、ここで前記バイナリノードの各々は下方ノードを有する左ブランチと下方ノードを有する右ブランチを有するとともに２つの前記下方ノードに対する２進演算を表わし、また前記リーフノードの各々は１つ以上の前記グラフィックオブジェクトが重なったグラフィックオブジェクトを表し、重なっているグラフィックオブジェクトの各々は左ノード領域、共通領域、および右ノード領域を含み、

前記表現ツリーを表すテーブルを生成し、ここで前記テーブルは関連するバイナリノードとリーフノードに対応する複数のレコードを含み、そして、前述の関連バイナリのノードのそれぞれの前記レコードは、前記左領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第１フィールドと、右領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第２フィールドと、前記関連するバイナリノードの左ブランチがアクティブであるかどうかを示す第３フィールドと、前記関連するバイナリノードの右ブランチがアクティブかどうかを示す第４フィールドとを備え、

１つ以上の画素位置のグループを決定し、

前記グループの各々に関して、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断し、

前記グループの各々に関して、前記判断されたアクティブ及び非アクティブのブランチのために、前記テーブルの前記第３及び第４フィールドを更新し、

それぞれの前記グループに関して、前記更新されたテーブルに従って、前記表現ツリーをなぞり、ここで、先になぞられたバイナリノードの左右ブランチがアクティブであるか、または、前記左ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの左ブランチがアクティブであり、かつ右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチがその下方ノードへとなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブであるか、または前記右ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチがその下方ノードへとなぞられ、

前記グループの各々に関して、前記右及び左ブランチがアクティブである先になぞられたバイナリノードに対して演算命令を生成し、なぞられたリーフノードに対してリーフ値命令を生成し、

前記グループの各々に関して、前記イメージを描画するために、前記生成された対応する命令を実行することを特徴とする方法。

【請求項１０】　前記テーブルは、前記関連バイナリノードのそれぞれの前記レコードに関して、前記関連バイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記関連バイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記関連バイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトを隠すかどうかを表す第５フィールドと、前記関連バイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトが、前記関連バイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記関連バイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトを隠すかどうかを表す第６フィールドとを更に備え、前記なぞるステップは、前記更新されたテーブルに従って、前記表現ツリーをなぞり、ここで先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合は、先になぞられたバイナリノードの左ブランチの下方ノードへ向かってなぞられ、先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合は、先になぞられたバイナリノードの右ブランチの下方ノードへ向かってなぞられることを特徴とする請求項９に記載の方法。

【請求項１１】　前記第５フィールド及び第６フィールドは、CLIP INもしくはCLIP OUT動作を実行するのに使用されることを特徴とする請求項１０に記載の方法。

【請求項１２】　有向隣接図表のための命令を生成する装置であって、前記有向隣接図表は１つ以上の親ノードと１つ以上のリーフノードを備え、それぞれの親ノードは演算子を表して、それぞれにぶらさがるブランチを有し、それぞれのリーフノードはグラフィックオブジェクトを表しており、前記装置が、

１つ以上の画素位置のグループを決定する手段と、

それぞれの前記グループのために、演算子のアクティビティに従って前記有向隣接図表の一部を決定する手段と、該有向隣接図表の該部分は、有向隣接図表へデータを渡す部分であり、

それぞれの前記グループのために、有向隣接図表の前記決定された部分への命令を生成する手段とを備え、ここで、演算子命令はアクティブなブランチを有する有向隣接図表の決定された部分のそれらの演算子のために生成され、リーフ命令は、１つ以上の画素位置の前記グループでアクティブなそれらのグラフィックオブジェクトのために生成されることを特徴とする装置。

【請求項１３】　演算子のアクティビティと親ノードのブランチのアクティビティに関するデータを格納するテーブルを生成する手段を備えることを特徴とする請求項１２に記載の装置。

【請求項１４】　前記決定する手段は、それぞれの前記

グループのために前記テーブルに格納された前記データを更新する手段を備えることを特徴とする請求項１３に記載の装置。

【請求項１５】　前記命令を生成する手段は、有向隣接図表の前記部分をなぞり、有向隣接図表の前記部分において、アクティブな演算子のための命令とリーフ値の命令とを生成する手段を含むことを特徴とする請求項１２に記載の装置。

【請求項１６】　前記有向隣接図表は表現ツリーであることを特徴とする請求項１２に記載の装置。

【請求項１７】　前記親ノードは２項演算子を表すことを特徴とする請求項１２に記載の装置。

【請求項１８】　表現ツリーを生成する装置であって、前記表現ツリーは、１つ以上のバイナリノードと複数のリーフノードを有する複数のノードを有し、ここで、それぞれの前記バイナリノードはそれにぶらさがる左ブランチと右ブランチとを有して該２つのぶらさがるノードへの２進演算を表し、それぞれの前記ノードは、１個以上の図形オブジェクトが重なった図形オブジェクトを表し、前記図形オブジェクトの重なりの各々が、左ノード領域、共通領域、および右ノード領域を含み、前記装置が、

１つ以上の画素位置のグループを決定する手段と、

それぞれの前記グループについて、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断する手段と、

それぞれの前記グループについて、前記表現ツリーをなぞる手段と、ここで、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記左ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの左ブランチがアクティブで右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチにぶらさがるノードへなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記右ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチにぶらさがるノードへなぞられ、

それぞれの前記グループのために、前記アクティブな左右ブランチを有する前記なぞられたバイナリノードのための演算子命令、およびなぞられたリーフノードのためのリーフ値命令を生成する手段とを備えることを特徴とする装置。

【請求項１９】　前記なぞる手段は表現ツリーをなぞり、前記先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリノードの下方ノードの共通領域において、前記先になぞられたバイナリノードの前記左ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合、前記先になぞられたバイナリノードの左ブランチの下方ノードへとなぞられ、また、先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの前記左ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリノードの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すオブジェクトを隠さない場合、前記先になぞられたバイナリノードの右ブランチの下方ノードへなぞられることを特徴とする請求項１８に記載の装置。

【請求項２０】　表現ツリーから、複数の走査線と各走査線に複数の画素位置を有するラスタ画素イメージを描画する装置であって、前記表現ツリーは複数のノードに１つ以上のバイナリノードと複数のリーフノードを有し、ここで前記バイナリノードの各々は下方ノードを有する左ブランチと下方ノードを有する右ブランチを有するとともに２つの前記下方ノードに対する２進演算を表わし、また前記リーフノードの各々は１つ以上の前記グラフィックオブジェクトが重なったグラフィックオブジェクトを表し、重なっているグラフィックオブジェクトの各々は左ノード領域、共通領域、および右ノード領域を含み、

前記表現ツリーを表すテーブルを生成する手段と、ここで前記テーブルは関連するバイナリノードとリーフノードに対応する複数のレコードを含み、そして、前述の関連バイナリのノードのそれぞれの前記レコードは、前記左領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第１フィールドと、右領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第２フィールドと、前記関連するバイナリノードの左ブランチがアクティブであるかどうかを示す第３フィールドと、前記関連するバイナリノードの右ブランチがアクティブかどうかを示す第４フィールドとを備え、

１つ以上の画素位置のグループを決定する手段と、

前記グループの各々に関して、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断する手段と、

前記グループの各々に関して、前記判断されたアクティブ及び非アクティブのブランチのために、前記テーブルの前記第３及び第４フィールドを更新する手段と、

それぞれの前記グループに関して、前記更新されたテーブルに従って、前記表現ツリーをなぞる手段と、ここで、先になぞられたバイナリノードの左右ブランチがアクティブであるか、または、前記左ノード領域が先にな

ぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの左ブランチがアクティブであり、かつ右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチがその下方ノードへとなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブであるか、または前記右ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチがその下方ノードへとなぞられ、
前記グループの各々に関して、前記右及び左ブランチがアクティブである先になぞられたバイナリノードに対して演算命令を生成し、なぞられたリーフノードに対してリーフ値命令を生成する手段と、
前記グループの各々に関して、前記イメージを描画するために、前記生成された対応する命令を実行する手段とを備えることを特徴とする装置。

【請求項２１】　前記テーブルは、前記関連バイナリノードのそれぞれの前記レコードに関して、前記関連バイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記関連バイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記関連バイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトを隠すかどうかを表す第５フィールドと、前記関連バイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトが、前記関連バイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記関連バイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトを隠すかどうかを表す第６フィールドとを更に備え、前記なぞる手段は、前記更新されたテーブルに従って、前記表現ツリーをなぞり、ここで先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合は、先になぞられたバイナリノードの左ブランチの下方ノードへ向かってなぞられ、先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合は、先になぞられたバイナリノードの右ブランチの下方ノードへ向かってなぞられることを特徴とする請求項２０に記載の装置。

【請求項２２】　前記第５フィールド及び第６フィールドは、CLIP INもしくはCLIP OUT動作を実行するのに使用されることを特徴とする請求項２１に記載の装置。

【請求項２３】　有向隣接図表のための命令を生成するためのコンピュータプログラムを備えたコンピュータ可読媒体であって、前記有向隣接図表は１つ以上の親ノードと１つ以上のリーフノードを備え、それぞれの親ノードは演算子を表して、それぞれにぶらさがるブランチを有し、それぞれのリーフノードはグラフィックオブジェクトを表しており、前記コンピュータプログラムが、
１つ以上の画素位置のグループを決定するためのコードと、
それぞれの前記グループのために、演算子のアクティビティに従って前記有向隣接図表の一部を決定するためのコードと、該有向隣接図表の該部分は、有向隣接図表へデータを渡す部分であり、
それぞれの前記グループのために、有向隣接図表の前記決定された部分への命令を生成するためのコードとを備え、ここで、演算子命令はアクティブなブランチを有する有向隣接図表の決定された部分のそれらの演算子のために生成され、リーフ命令は、１つ以上の画素位置の前記グループでアクティブなそれらのグラフィックオブジェクトのために生成されることを特徴とするコンピュータ可読媒体。

【請求項２４】　前記コンピュータプログラムが、演算子のアクティビティと親ノードのブランチのアクティビティに関するデータを格納するテーブルを生成するためのコードを備えることを特徴とする請求項２３に記載のコンピュータ可読媒体。

【請求項２５】　前記決定するためのコードは、それぞれの前記グループのために前記テーブルに格納された前記データを更新するためのコードを備えることを特徴とする請求項２４に記載のコンピュータ可読媒体。

【請求項２６】　前記命令を生成するためのコードは、有向隣接図表の前記部分をなぞり、有向隣接図表の前記部分において、アクティブな演算子のための命令とリーフ値の命令とを生成するためのコードを含むことを特徴とする請求項２３に記載のコンピュータ可読媒体。

【請求項２７】　前記有向隣接図表は表現ツリーであることを特徴とする請求項２３に記載のコンピュータ可読媒体。

【請求項２８】　前記親ノードは２項演算子を表すことを特徴とする請求項２３に記載のコンピュータ可読媒体。

【請求項２９】　表現ツリーを生成するためのコンピュータプログラムを備えるコンピュータ可読媒体であって、前記表現ツリーは、１つ以上のバイナリノードと複数のリーフノードを有する複数のノードを有し、ここで、それぞれの前記バイナリノードはそれにぶらさがる

左ブランチと右ブランチとを有して該２つのぶらさがるノードへの２進演算を表し、それぞれの前記ノードは、１個以上の図形オブジェクトが重なった図形オブジェクトを表し、前記図形オブジェクトの重なりの各々が、左ノード領域、共通領域、および右ノード領域を含み、前記コンピュータプログラムが、

１つ以上の画素位置のグループを決定するためのコードと、

それぞれの前記グループについて、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断するためのコードと、

それぞれの前記グループについて、前記表現ツリーをなぞるためのコードと、ここで、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記左ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの左ブランチがアクティブで右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチにぶらさがるノードへなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記右ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチにぶらさがるノードへなぞられ、

それぞれの前記グループのために、前記アクティブな左右ブランチを有する前記なぞられたバイナリノードのための演算子命令、およびなぞられたリーフノードのためのリーフ値命令を生成するためのコードとを備えること特徴とするコンピュータ可読媒体。

【請求項３０】　前記なぞるためのコードは表現ツリーをなぞり、前記先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリノードの下方ノードの共通領域において、前記先になぞられたバイナリノードの前記左ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合、前記先になぞられたバイナリノードの左ブランチの下方ノードへとなぞられ、また、先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの前記左ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリノードの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すオブジェクトを隠さない場合、前記先になぞられたあらゆるバイナリノードの右ブランチの下方ノードへなぞられることを特徴とする請求項２９に記載のコンピュータ可読媒体。

【請求項３１】　表現ツリーから、複数の走査線と各走査線に複数の画素位置を有するラスタ画素イメージを描画するためのコンピュータプログラムを備えるコンピュータ可読媒体であって、前記表現ツリーは複数のノードに１つ以上のバイナリノードと複数のリーフノードを有し、ここで前記バイナリノードの各々は下方ノードを有する左ブランチと下方ノードを有する右ブランチを有するとともに２つの前記下方ノードに対する２進演算を表わし、また前記リーフノードの各々は１つ以上の前記グラフィックオブジェクトが重なったグラフィックオブジェクトを表し、重なっているグラフィックオブジェクトの各々は左ノード領域、共通領域、および右ノード領域を含み、

前記表現ツリーを表すテーブルを生成するためのコードと、ここで前記テーブルは関連するバイナリノードとリーフノードに対応する複数のレコードを含み、そして、前述の関連バイナリのノードのそれぞれの前記レコードは、前記左領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第１フィールドと、右領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第２フィールドと、前記関連するバイナリノードの左ブランチがアクティブであるかどうかを示す第３フィールドと、前記関連するバイナリノードの右ブランチがアクティブかどうかを示す第４フィールドとを備え、

１つ以上の画素位置のグループを決定するためのコードと、

前記グループの各々に関して、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断するためのコードと、

前記グループの各々に関して、前記判断されたアクティブ及び非アクティブのブランチのために、前記テーブルの前記第３及び第４フィールドを更新するためのコードと、

それぞれの前記グループに関して、前記更新されたテーブルに従って、前記表現ツリーをなぞるためのコードと、ここで、先になぞられたバイナリノードの左右ブランチがアクティブであるか、または、前記左ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの左ブランチがアクティブであり、かつ右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチがその下方ノードへとなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブであるか、または前記右ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチがその下方ノードへとなぞられ、

前記グループの各々に関して、前記右及び左ブランチがアクティブである先になぞられたバイナリノードに対して演算命令を生成し、なぞられたリーフノードに対してリーフ値命令を生成するためのコードと、
前記グループの各々に関して、前記イメージを描画するために、前記生成された対応する命令を実行するためのコードとを備えることを特徴とするコンピュータ可読媒体。
【請求項３２】　前記テーブルは、前記関連バイナリノードのそれぞれの前記レコードに関して、前記関連バイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記関連バイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記関連バイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトを隠すかどうかを表す第５フィールドと、前記関連バイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトが、前記関連バイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記関連バイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトを隠すかどうかを表す第６フィールドとを更に備え、前記なぞるためのコードは、前記更新されたテーブルに従って、前記表現ツリーをなぞり、ここで先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合は、先になぞられたバイナリノードの左ブランチの下方ノードへ向かってなぞられ、先になぞられたバイナリノードの左右ブランチがアクティブであり、前記先になぞられたバイナリノードの左ブランチの下方ノードを表すグラフィックオブジェクトが、前記先になぞられたバイナリオブジェクトの下方ノードを表すグラフィックオブジェクトの共通領域において、前記先になぞられたバイナリノードの右ブランチの下方ノードを表すグラフィックオブジェクトを隠さない場合は、先になぞられたバイナリノードの右ブランチの下方ノードへ向かってなぞられることを特徴とする請求項３１に記載のコンピュータ可読媒体。
【請求項３３】　前記第５フィールド及び第６フィールドは、CLIP INもしくはCLIP OUT動作を実行するのに使用されることを特徴とする請求項３２に記載のコンピュータ可読媒体。
【発明の詳細な説明】
【０００１】著作権表示本特許明細書は著作権保護を受ける対象となる内容を含む。著作権所有者は、この特許明細書、もしくは特許庁からの関連する資料をレビューの目的で複製することに異議を唱えるものではないが、他の点においてはあらゆる著作権を保有するものである。
【０００２】発明の技術分野本発明は、グラフィックオブジェクトベースのイメージの描画に関するものである。特に、本発明は表現ツリーのための命令を生成するための方法と装置に関連する。また、本発明はラスタ画素イメージに表現ツリーを描画するための方法と装置に関連する。また、発明は上述の方法のいずれかを実行するためのコンピュータプログラムを備えたコンピュータ可読媒体に関連する。
【０００３】発明の背景
大部分のオブジェクトベースのグラフィックスシステムでは、ページまたは画面上の画素ベース画像を保持するためにフレームストアまたはページバッファが使用される。通常、グラフィックオブジェクトの輪郭は、計算され、塗り潰され、フレームストアに書き込まれる。二次元グラフィックスの場合、他のオブジェクトの手前にあるオブジェクトは、単純に背景オブジェクトを書き込んだ後にフレームストアに書き込まれ、これによって、画素単位で背景を置換する。これは、当技術分野では、「ペインタのアルゴリズム（Painter's algorithm）」として一般に知られているものである。オブジェクトは、最も奥のオブジェクトから最も手前のオブジェクトという優先順位で考慮され、通常は、各オブジェクトがスキャンラインの順序でラスタ化され、各スキャンラインに沿ったシーケンシャルな並びで画素がフレームストアに書き込まれる。
【０００４】この技法には、基本的に２つの問題がある。第１の問題は、フレームストア内のすべての画素への高速なランダムアクセスが必要であることである。これは、新たに考慮される各オブジェクトが、フレームストア内のどの画素にも影響を及ぼす可能性があるからである。このため、フレームストアは、通常は半導体のランダムアクセスメモリ（ＲＡＭ）内に保持される。高解像度カラープリンタの場合、必要なＲＡＭの量は非常に多くなり、通常は１００Ｍバイトを超える。このため、コストがかかり、高速での動作が困難である。第２の問題は、多数の画素がペイント（レンダリング）され、その後、時間的に後のオブジェクトによって上塗り（再レンダリング）されることである。このため、時間的に前のオブジェクトによる画素のペイントは、時間の浪費になる。
【０００５】大量のフレームストアの問題を克服するための方法の１つが、「バンディング（banding）」の使用である。バンディングを使用する場合には、フレームストアの一部だけが常時メモリ内に存在することになる。描画されるべきオブジェクトのすべてが「表示リスト」に保持される。画像全体は上記と同様にレンダリングされるが、存在するフレームストアの部分の外側をペ

イント（レンダリング）しようとする画素ペイント（レンダリング）動作は、「クリップ」アウトされる。オブジェクトのすべてが描画された後、フレームストアの部分をプリンタ（または他の場所）に送り、フレームストアの別の部分を選択し、この処理を繰り返す。この技法には、ペナルティが伴う。たとえば、描画されるオブジェクトは考慮され、何度も（バンドごとに1回）再考慮されなければならない。バンドの数が増えるにつれて、レンダリングが必要なオブジェクトの検査の繰り返し回数も増える。従って、バンディングの技法は、上塗りのコストという問題を解決しない。

【0006】いくつかの他のグラフィックシステムでは、スキャンラインの順で画像が検討される。やはり、描画されるオブジェクトのすべてが、表示リストに保存される。各スキャンライン上で、そのスキャンラインと交差するオブジェクトが優先順位の順でオブジェクトごとに検討され、オブジェクトのエッジの交差点の間の画素のスパンが、ラインストアにセットされる。この技法も、大量のフレームストアの問題を克服するが、やはり上塗りの問題をこうむる。

【0007】これらのほかに、大きいフレームストアの問題と上塗りの問題の両方を解決する技法がある。そのような技法の1つでは、各スキャンラインが順番に作られる。やはり、描画されるオブジェクトのすべてが、表示リストに保存される。各スキャンラインでは、そのスキャンラインと交差するオブジェクトのエッジが、スキャンラインとの交差の座標の昇順で保持される。これらの交差の点、すなわちエッジの交点が、順番に検討され、アクティブフィールドの配列のトグルに使用される。そのスキャンライン上での対象となるオブジェクト優先順位ごとに1つのアクティブフィールドがある。考慮されるエッジの対のそれぞれの間で、最初のエッジと次のエッジの間にある各画素の色データが、アクティブフラグに対する優先順位エンコーダを使用してどの優先順位が最も上にあるかを判定すること、および2つのエッジの間のスパンの画素に関するその優先順位に関連する色を使用することによって生成される。次のスキャンラインに備えて、各エッジの交差の座標が、各エッジの性質に応じて更新される。この更新の結果として誤ってソートされた状態になった隣接するエッジは、交換される。新しいエッジも、エッジのリストに合併される。

【0008】この技法は、フレームストアまたはラインストアがなく、上塗りがなく、オーダーNの回数（このNは優先順位の数）ではなく一定のオーダーの時間でオブジェクト優先順位が処理されるという大きい長所を有する。

【0009】しかしながら、これらの優先順位エンコーダはツリーベースのグラフィカルな表現に対処するための柔軟性を欠いている。特に、Porter,T、Duff,Tによる「合成ディジタルイメージ（Compositing Digital Images）」（Vol.18、No.3(1984)pp.253-259）で記述されるような、OUTとATOP PorterおよびDuffの合成演算子を含むそれらの表現に対処するための柔軟性を欠く。与えられた画素位置において、合成演算が、その演算によって合成されたグラフィックオブジェクトのどれが有効であるのかに依存するために、問題が生じる。この問題の1つの解決は切取りオブジェクトの複雑な整理を使用することである。それは多くの余分なエッジ処理を必要として、切取りオブジェクトに関する多くのレベルを必要とする。

【0010】本発明の目的は、既存の構成の1つまたは複数の短所を実質的に克服するか、少なくとも改善することである。

【0011】発明の要約

本発明の第1の態様によれば次の方法が提供される。すなわち、有向隣接図表のための命令を生成する方法であって、前記有向隣接図表は1つ以上の親ノードと1つ以上のリーフノードを備え、それぞれの親ノードは演算子を表して、それぞれにぶらさがるブランチを有し、それぞれのリーフノードはグラフィックオブジェクトを表しており、前記方法が、1つ以上の画素位置のグループを決定し、それぞれの前記グループのために、演算子のアクティビティに従って前記有向隣接図表の一部を決定し、該有向隣接図表の該部分は、有向隣接図表へデータを渡す部分であり、それぞれの前記グループのために、有向隣接図表の前記決定された部分への命令を生成し、ここで、演算子命令はアクティブなブランチを有する有向隣接図表の決定された部分のそれらの演算子のために生成され、リーフ命令は、1つ以上の画素位置の前記グループでアクティブなそれらのグラフィックオブジェクトのために生成される。

【0012】また、本発明の第2の態様によれば次の方法が提供される。すなわち、表現ツリーを生成する方法であって、前記表現ツリーは、1つ以上のバイナリノードと複数のリーフノードを有する複数のノードを有し、ここで、それぞれの前記バイナリノードはそれにぶらさがる左ブランチと右ブランチとを有して該2つのぶらさがるノードへの2進演算を表し、それぞれの前記ノードは、1個以上の図形オブジェクトが重なった図形オブジェクトを表し、前記図形オブジェクトの重なりの各々が、左ノード領域、共通領域、および右ノード領域を含み、前記方法が、1つ以上の画素位置のグループを決定し、それぞれの前記グループについて、前記1つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断し、それぞれの前記グループについて、前記表現ツリーをなぞり、ここで、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記左ノード領域が前記先になぞられたバイナリノードの2進演算に必要であり、前記先になぞられたバイナリノードの左ブランチがアクティ

ブで右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチにぶらさがるノードへなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記右ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチにぶらさがるノードへなぞられ、それぞれの前記グループのために、前記アクティブな左右ブランチを有する前記なぞられたバイナリノードのための演算子命令、およびなぞられたリーフノードのためのリーフ値命令を生成する。

【００１３】また、本発明の第３の態様によれば次の方法が提供される。すなわち、表現ツリーから、複数の走査線と各走査線に複数の画素位置を有するラスタ画素イメージを描画する方法であって、前記表現ツリーは複数のノードに１つ以上のバイナリノードと複数のリーフノードを有し、ここで前記バイナリノードの各々は下方ノードを有する左ブランチと下方ノードを有する右ブランチを有するとともに２つの前記下方ノードに対する２進演算を表わし、また前記リーフノードの各々は１つ以上の前記グラフィックオブジェクトが重なったグラフィックオブジェクトを表し、重なっているグラフィックオブジェクトの各々は左ノード領域、共通領域、および右ノード領域を含み、前記表現ツリーを表すテーブルを生成し、ここで前記テーブルは関連するバイナリノードとリーフノードに対応する複数のレコードを含み、そして、前述の関連バイナリのノードのそれぞれの前記レコードは、前記左領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第１フィールドと、右領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第２フィールドと、前記関連するバイナリノードの左ブランチがアクティブであるかどうかを示す第３フィールドと、前記関連するバイナリノードの右ブランチがアクティブかどうかを示す第４フィールドとを備え、１つ以上の画素位置のグループを決定し、前記グループの各々に関して、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断し、前記グループの各々に関して、前記判断されたアクティブ及び非アクティブのブランチのために、前記テーブルの前記第３及び第４フィールドを更新し、それぞれの前記グループに関して、前記更新されたテーブルに従って、前記表現ツリーをなぞり、ここで、先になぞられたバイナリノードの左右ブランチがアクティブであるか、または、前記左ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの左ブランチがアクティブであり、かつ右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチがその下方ノードへとなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブであるか、または前記右ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチがその下方ノードへとなぞられ、前記グループの各々に関して、前記右及び左ブランチがアクティブである先になぞられたバイナリノードに対して演算命令を生成し、なぞられたリーフノードに対してリーフ値命令を生成し、前記グループの各々に関して、前記イメージを描画するために、前記生成された対応する命令を実行する。

【００１４】また本発明の第４の態様によれば、次の装置が提供される。すなわち、有向隣接図表のための命令を生成する装置であって、前記有向隣接図表は１つ以上の親ノードと１つ以上のリーフノードを備え、それぞれの親ノードは演算子を表して、それぞれにぶらさがるブランチを有し、それぞれのリーフノードはグラフィックオブジェクトを表しており、前記装置が、１つ以上の画素位置のグループを決定する手段と、それぞれの前記グループのために、演算子のアクティビティに従って前記有向隣接図表の一部を決定する手段と、該有向隣接図表の該部分は、有向隣接図表へデータを渡す部分であり、それぞれの前記グループのために、有向隣接図表の前記決定された部分への命令を生成する手段とを備え、ここで、演算子命令はアクティブなブランチを有する有向隣接図表の決定された部分のそれらの演算子のために生成され、リーフ命令は、１つ以上の画素位置の前記グループでアクティブなそれらのグラフィックオブジェクトのために生成される。

【００１５】また本発明の第５の態様によれば、次の装置が提供される。すなわち、表現ツリーを生成する装置であって、前記表現ツリーは、１つ以上のバイナリノードと複数のリーフノードを有する複数のノードを有し、ここで、それぞれの前記バイナリノードはそれにぶらさがる左ブランチと右ブランチとを有して該２つのぶらさがるノードへの２進演算を表し、それぞれの前記ノードは、１個以上の図形オブジェクトが重なった図形オブジェクトを表し、前記図形オブジェクトの重なりの各々が、左ノード領域、共通領域、および右ノード領域を含み、前記装置が、１つ以上の画素位置のグループを決定する手段と、それぞれの前記グループについて、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断する手段と、それぞれの前記グループについて、前記表現ツリーをなぞる手段と、ここで、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記左ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノード

の左ブランチがアクティブで右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチにぶらさがるノードへなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記右ノード領域が前記先になぞられたバイナリノードの2進演算に必要であり、前記先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチにぶらさがるノードへなぞられ、それぞれの前記グループのために、前記アクティブな左右ブランチを有する前記なぞられたバイナリノードのための演算子命令、およびなぞられたリーフノードのためのリーフ値命令を生成する手段とを備える。

【0016】また本発明の第6の態様によれば、次の装置が提供される。すなわち、表現ツリーから、複数の走査線と各走査線に複数の画素位置を有するラスタ画素イメージを描画する装置であって、前記表現ツリーは複数のノードに1つ以上のバイナリノードと複数のリーフノードを有し、ここで前記バイナリノードの各々は下方ノードを有する左ブランチと下方ノードを有する右ブランチを有するとともに2つの前記下方ノードに対する2進演算を表わし、また前記リーフノードの各々は1つ以上の前記グラフィックオブジェクトが重なったグラフィックオブジェクトを表し、重なっているグラフィックオブジェクトの各々は左ノード領域、共通領域、および右ノード領域を含み、前記表現ツリーを表すテーブルを生成する手段と、ここで前記テーブルは関連するバイナリノードとリーフノードに対応する複数のレコードを含み、そして、前述の関連バイナリのノードのそれぞれの前記レコードは、前記左領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第1フィールドと、右領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第2フィールドと、前記関連するバイナリノードの左ブランチがアクティブであるかどうかを示す第3フィールドと、前記関連するバイナリノードの右ブランチがアクティブかどうかを示す第4フィールドとを備え、1つ以上の画素位置のグループを決定する手段と、前記グループの各々に関して、前記1つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断する手段と、前記グループの各々に関して、前記判断されたアクティブ及び非アクティブのブランチのために、前記テーブルの前記第3及び第4フィールドを更新する手段と、それぞれの前記グループに関して、前記更新されたテーブルに従って、前記表現ツリーをなぞる手段と、ここで、先になぞられたバイナリノードの左右ブランチがアクティブであるか、または、前記左ノード領域が先になぞられたバイナリノードの2進演算に必要であり、該先になぞられたバイナリノードの左ブランチがアクティブであり、かつ右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチがその下方ノードへとなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブであるか、または前記右ノード領域が先になぞられたバイナリノードの2進演算に必要であり、該先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチがその下方ノードへとなぞられ、前記グループの各々に関して、前記右及び左ブランチがアクティブである先になぞられたバイナリノードに対して演算命令を生成し、なぞられたリーフノードに対してリーフ値命令を生成する手段と、前記グループの各々に関して、前記イメージを描画するために、前記生成された対応する命令を実行する手段とを備える。

【0017】また本発明の第7の態様によれば、次のコンピュータ可読媒体が提供される。すなわち、有向隣接図表のための命令を生成するためのコンピュータプログラムを備えたコンピュータ可読媒体であって、前記有向隣接図表は1つ以上の親ノードと1つ以上のリーフノードを備え、それぞれの親ノードは演算子を表して、それぞれにぶらさがるブランチを有し、それぞれのリーフノードはグラフィックオブジェクトを表しており、前記コンピュータプログラムが、1つ以上の画素位置のグループを決定するためのコードと、それぞれの前記グループのために、演算子のアクティビティに従って前記有向隣接図表の一部を決定するためのコードと、該有向隣接図表の該部分は、有向隣接図表へデータを渡す部分であり、それぞれの前記グループのために、有向隣接図表の前記決定された部分への命令を生成するためのコードとを備え、ここで、演算子命令はアクティブなブランチを有する有向隣接図表の決定された部分のそれらの演算子のために生成され、リーフ命令は、1つ以上の画素位置の前記グループでアクティブなそれらのグラフィックオブジェクトのために生成される。

【0018】また本発明の第8の態様によれば、次のコンピュータ可読媒体が提供される。すなわち、表現ツリーを生成するためのコンピュータプログラムを備えるコンピュータ可読媒体であって、前記表現ツリーは、1つ以上のバイナリノードと複数のリーフノードを有する複数のノードを有し、ここで、それぞれの前記バイナリノードはそれにぶらさがる左ブランチと右ブランチとを有して該2つのぶらさがるノードへの2進演算を表し、それぞれの前記ノードは、1個以上の図形オブジェクトが重なった図形オブジェクトを表し、前記図形オブジェクトの重なりの各々が、左ノード領域、共通領域、および右ノード領域を含み、前記コンピュータプログラムが、1つ以上の画素位置のグループを決定するためのコードと、それぞれの前記グループについて、前記1つ以上のバイナリノードの左右のブランチがアクティブであるか

非アクティブであるかを判断するためのコードと、それぞれの前記グループについて、前記表現ツリーをなぞるためのコードと、ここで、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記左ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの左ブランチがアクティブで右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチにぶらさがるノードへなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブである場合、或いは、前記右ノード領域が前記先になぞられたバイナリノードの２進演算に必要であり、前記先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチにぶらさがるノードへなぞられ、それぞれの前記グループのために、前記アクティブな左右ブランチを有する前記なぞられたバイナリノードのための演算子命令、およびなぞられたリーフノードのためのリーフ値命令を生成するためのコードとを備える。

【００１９】また本発明の第９の態様によれば、次のコンピュータ可読媒体が提供される。すなわち、表現ツリーから、複数の走査線と各走査線に複数の画素位置を有するラスタ画素イメージを描画するためのコンピュータプログラムを備えるコンピュータ可読媒体であって、前記表現ツリーは複数のノードに１つ以上のバイナリノードと複数のリーフノードを有し、ここで前記バイナリノードの各々は下方ノードを有する左ブランチと下方ノードを有する右ブランチを有するとともに２つの前記下方ノードに対する２進演算を表わし、また前記リーフノードの各々は１つ以上の前記グラフィックオブジェクトが重なったグラフィックオブジェクトを表し、重なっているグラフィックオブジェクトの各々は左ノード領域、共通領域、および右ノード領域を含み、前記表現ツリーを表すテーブルを生成するためのコードと、ここで前記テーブルは関連するバイナリノードとリーフノードに対応する複数のレコードを含み、そして、前述の関連バイナリのノードのそれぞれの前記レコードは、前記左領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第１フィールドと、右領域が前記バイナリノードに関連する演算子の動作に必要であるかどうかを示す第２フィールドと、前記関連するバイナリノードの左ブランチがアクティブであるかどうかを示す第３フィールドと、前記関連するバイナリノードの右ブランチがアクティブかどうかを示す第４フィールドとを備え、１つ以上の画素位置のグループを決定するためのコードと、前記グループの各々に関して、前記１つ以上のバイナリノードの左右のブランチがアクティブであるか非アクティブであるかを判断するためのコードと、前記グループの各々に関して、前記判断されたアクティブ及び非アクティブのブランチのために、前記テーブルの前記第３及び第４フィールドを更新するためのコードと、それぞれの前記グループに関して、前記更新されたテーブルに従って、前記表現ツリーをなぞるためのコードと、ここで、先になぞられたバイナリノードの左右ブランチがアクティブであるか、または、前記左ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの左ブランチがアクティブであり、かつ右ブランチが非アクティブである場合に、前記先になぞられたバイナリノードの左ブランチがその下方ノードへとなぞられ、また、前記先になぞられたバイナリノードの左右ブランチがアクティブであるか、または前記右ノード領域が先になぞられたバイナリノードの２進演算に必要であり、該先になぞられたバイナリノードの右ブランチがアクティブであり、左ブランチが非アクティブである場合に、先になぞられたバイナリノードの右ブランチがその下方ノードへとなぞられ、前記グループの各々に関して、前記右及び左ブランチがアクティブである先になぞられたバイナリノードに対して演算命令を生成し、なぞられたリーフノードに対してリーフ値命令を生成するためのコードと、前記グループの各々に関して、前記イメージを描画するために、前記生成された対応する命令を実行するためのコードとを備える。

【００２０】発明の実施の形態

添付の１つ又は複数の図面において同一の参照番号が付されたステップ及び／或いは機能は、特に断りのない限り、同じ機能或いは動作を記述するものである。

【００２１】図１は、コンピュータグラフィックオブジェクト画像のレンダリングおよびプレゼンテーションのために構成されたコンピュータシステム１を概略的に示す図である。このシステムには、システムランダムアクセスメモリ（ＲＡＭ）３に関連するホストプロセッサ２が含まれ、システムＲＡＭ３には、不揮発性のハードディスクドライブ５または類似の装置と、揮発性の半導体ＲＡＭ４を含めることができる。システム１には、システム読取専用メモリ（ＲＯＭ）６も含まれ、システムＲＯＭ６は、通常は半導体ＲＯＭ７をベースとし、多くの場合、コンパクトディスク装置（ＣＤ－ＲＯＭ）８によって補足することができる。システム１には、ラスタ式に動作するビデオ表示装置（ＶＤＵ）またはプリンタなどの、画像を表示するための手段１０も組み込むことができる。

【００２２】システム１に関して上で説明した構成要素は、バスシステム９を介して相互接続され、IBM PC/ATタイプのパーソナルコンピュータおよびそれから発展した構成、Sun Sparcstationsおよびその類似装置など、当技術分野で周知のコンピュータシステムの通常動作モードにて動作可能である。

【００２３】また、図１に図示されているように、画素

シーケンシャルレンダリング装置２０がバス９に接続される。画素シーケンシャルレンダリング装置２０は、好ましい実施形態において、システム１からバス９を介して命令およびデータを供給され、グラフィックオブジェクトベースの記述から導出される画素ベースの画像のシーケンシャルレンダリングを実行する。レンダリング装置２０は、オブジェクト記述のレンダリングのためにシステムＲＡＭ３を使用することができるが、レンダリング装置２０は、通常は半導体ＲＡＭから形成される、専用のレンダリング格納部３０に関連付けられることが好ましい。

【００２４】本発明の大綱は、有向隣接図表に対する命令を生成すること、具体的には表現ツリーを生成する際のアプリケーションを有する。これは、好適な実施形態においては、画素シーケンシャルレンダリング装置２０のアクティビティ判断及び命令生成モジュール５００（図５）において実現される。このモジュールは、本明細書のセクション「３．０　アクティビティ判断及び命令生成モジュール」において詳細に説明される。

【００２５】次に図２を参照すると、好ましい実施形態の全体的な機能のデータ流れ図が示されている。図２の機能流れ図は、オブジェクトグラフィック記述１１から始まる。このオブジェクトグラフィック記述１１は、ホストプロセッサ２によって作り出されるグラフィックオブジェクトに適切な様式で、且つ/又は、システムＲＡＭ３内に記憶されたりシステムＲＯＭ６から導出されるグラフィックオブジェクトに適切な様式でそれらのパラメータを記述するのに使用される。そして、このグラフィックオブジェクト記述は、画素ベースの画像をレンダリングするために、画素シーケンシャルレンダリング装置２０によって解釈され得る。たとえば、オブジェクトグラフィック記述１１は、ディスプレイ上の１点から別の点までをなぞる直線のエッジ（単純ベクトル）または、直交する線を含む複数のエッジによって二次元オブジェクトが定義される直交エッジフォーマットを含むいくつかのフォーマットによって、エッジを有するオブジェクトを組み込むことができる。これ以外に、連続曲線によってオブジェクトが定義されるフォーマットも適当であり、これらには、乗算を実行する必要なしに二次曲線を単一の出力空間内でレンダリングできるようにする複数のパラメータによって単一の曲線を記述できる二次多項式の線分を含めることができる。それ以外の、三次スプラインやその類似物などのデータフォーマットを使用することもできる。オブジェクトには、多数の異なるエッジタイプの混合物を含めることができる。通常、すべてのフォーマットに共通するのは、それぞれの線（直線であれ曲線であれ）の始点と終点との識別子であり、通常は、これらは、スキャンライン番号によって識別され、したがって、その曲線をレンダリングすることのできる特定の出力空間が定義される。

【００２６】たとえば、図１４Ａに、線分を適当に記述し、レンダリングするために、２つの線分６０１および６０２に分割する必要があるエッジ６００に対する従来のエッジ記述を示す。分割の必要が生じるのは、従来のエッジ記述が、二次式を介して簡単に計算されるが、変曲点６０４に適応することができないからである。したがって、エッジ６００は、終点６０３および６０４または終点６０４および６０５を有する２つの別々のエッジとして扱われた。図１４Ｂに、終点６１１および６１２と制御点６１３および６１４によって記述される三次スプライン６１０を示す。このフォーマットでは、レンダリングのために三次多項式の計算が必要であり、したがって、計算時間がかかる。

【００２７】図１４Ｃ及び図１４Ｄに、好ましい実施形態に適用可能なエッジの例を示す。好ましい実施形態では、エッジは、単一の実体とみなされ、必要であれば、異なるフォーマットで記述できるエッジの部分を示すために区分されるが、その具体的な目的は、各部分の記述の複雑さが最小限になるようにすることである。

【００２８】図１４Ｃの左側には、スキャンラインＡ～Ｍの間にまたがる単一のエッジ６２０が示されている。エッジは、start_x、start_y、エッジの次の線分を指すアドレスを含む１つまたは複数の線分記述、および、エッジの終了に使用される最終線分を含む複数のパラメータによって記述される。好ましい実施形態によれば、エッジ６２０は、３つのステップ線分、１つのベクトル線分および１つの二次線分として記述することができる。ステップ線分は、単純に、ｘステップ値とｙステップ値を有するものとして定義される。図示の３つのステップ線分の場合、線分記述は［０，２］、［＋２，２］および［＋２，０］である。なお、ｘステップ値は符号付きであり、これによってステップの向きが示されるが、ｙステップ値は、必ず、スキャンラインの値が増えるラスタスキャン方向であるから符号なしである。次の線分はベクトル線分であり、通常はパラメータstart_x、start_y、finish_yおよび傾斜（DX）を必要とする。この例では、ベクトル線分がエッジ６２０の中間線分であるから、start_xおよびstart_yは、前の線分から生じるので、省略することができる。傾斜値（DX）は、符号付きであり、前のスキャンラインのｘ値に加算されて、現行スキャンラインのｘ値を与える。図示の例では、DX＝＋１である。次の線分は、二次線分であり、これは、ベクトル線分に対応する構造を有するが、さらに、やはり符号付きであり、線分の傾斜を変更するためにDXに加算される２階値（DDX）も有する。

【００２９】図１４Ｄの右側の線分は、好ましい実施形態による三次曲線の例を示す図であり、これには上記の二次線分に対応する記述が含まれるが、線分の傾斜の変化の割合を変更するためにDDXに加算される符号付きの３階値（DDDX）が追加されている。他の多数階について

も、同様にして実施することができる。

【００３０】上記から、エッジの線分を記述する複数のデータフォーマットを処理する能力があると、複雑で計算コストの高い数学演算に頼らずに、エッジの記述と評価を単純化することができることが明白である。これに対して、図１４Ａの従来技術のシステムでは、直交、ベクトルまたは二次のいずれであれ、すべてのエッジを二次形式で記述する必要があった。

【００３１】好ましい実施形態の動作を、図８に示された画像７８のレンダリングという単純な例に関して説明する。画像７８は、２つのグラフィカルオブジェクト、具体的に言うと、不透明の赤色の長方形９０とその上にレンダリングされ、これによって長方形９０を部分的に隠す、部分的に透明の青色の三角形８０を含む。図からわかるように、長方形９０には、種々の画素位置（Ｘ）とスキャンライン位置（Ｙ）の間で定義された辺（エッジ）９２、９４、９６および９８が含まれる。エッジ９６および９８は、スキャンライン上に形成される（したがってこれらと平行である）ので、長方形９０の実際のオブジェクト記述は、図９Ａに示されているように、エッジ９２および９４だけに基づくものとすることができる。図９Ａにおいて、エッジ９２は、画素位置（４０，３５）から始まり、ラスタ方向で画面の下側へ延びて、画素位置（４０，１０５）で終わる。同様に、エッジ９４は、画素位置（１６０，３５）から位置（１６０，１０５）まで延びる。長方形グラフィックオブジェクト９０の水平部分は、単にエッジ９２からエッジ９４へラスタ化された形で走査することによって得ることができる。

【００３２】しかし、青い三角形のオブジェクト８０は、３つのオブジェクトエッジ８２、８４および８６によって定義され、各エッジは、三角形の頂点を定義するベクトルとみなされる。エッジ８２および８４は、画素位置（１００，２０）から始まり、それぞれ画素位置（１７０，９０）または（３０，９０）まで延びる。エッジ８６は、これら２つの画素位置の間で、従来のラスタ化された左から右への方向に延びる。この特定の例では、エッジ８６が、上で述べたエッジ９６および９８と同様に水平なので、エッジ８６が定義されることは必須ではない。というのは、エッジ８６が、エッジ８２および８４の終点をもつという特徴があるからである。エッジ８２および８４の記述に使用される始点および終点の画素位置のほかに、これらのエッジのそれぞれに、この場合ではそれぞれ＋１または－１の傾斜値が関連付けられる。

【００３３】図１０に、スキャンライン３５で始まる長方形９０がレンダリングされる様子と、エッジ８２および８４がスキャンライン３５とどのように交差するかを示す。図１０から、画像７８のラスタ化には、高い優先順位レベルを有するオブジェクトが、低い優先順位レベルを有するオブジェクトの「上」にレンダリングされる形で２つのオブジェクト９０および８０が描画される必要のあることがわかる。これを図１１に示す。図１１は、画像７８のレンダリングに使用されるエッジリストレコードを示す図である。図１１のレコードには、オブジェクトごとに１つずつの、２つの項目が含まれる。これらの項目は、それぞれのオブジェクトのラスタレンダリング順での始点に対応するスキャンライン値で配置される。図１１から、エッジレコードのそれぞれが、オブジェクトに対応する優先順位レベルと、記述されるエッジの性質に関する詳細（たとえば色、傾斜など）を有することがわかる。

【００３４】再び図２に戻って説明する。レンダリングされるグラフィックオブジェクトの記述に必要なデータを識別したので、グラフィックシステム１は、表示リスト生成ステップ１２を実行する。

【００３５】表示リスト生成１２は、ＲＯＭ６およびＲＡＭ３を有するホストプロセッサ２上で実行されるソフトウェアモジュールとして実施されることが好ましい。表示リスト生成１２では、周知のグラフィック記述言語、グラフィックライブラリ呼出しまたは他のアプリケーション固有フォーマットのうちの１つまたは複数で表現されたオブジェクトグラフィック記述を表示リストに変換する。表示リストは、通常は、表示リスト格納部１３に書き込まれる。表示リスト格納部１３は、一般にＲＡＭ４内で形成されるが、レンダリング格納部３０内に形成するようにしてもよい。図３からわかるように、表示リスト格納部１３には、複数の構成要素を含めることができ、その１つは命令ストリーム１４であり、もう１つはエッジ情報１５であり、ラスタ画像画素データ１６を含めることができる。

【００３６】命令ストリーム１４には、特定の画像内で所望される特定のグラフィックオブジェクトをレンダリングするために画素シーケンシャルレンダリング装置２０によって読み取られる、命令として解釈可能なコードが含まれる。図８に示された画像の例では、命令ストリーム１４が、下記の形になり得る。

【００３７】（１）スキャンライン２０までレンダリング（何もしない）、（２）スキャンライン２０で２つの青いエッジ８２および８４を追加、（３）スキャンライン３５までレンダリング、（４）スキャンライン３５で２つの赤いエッジ９２および９４を追加、（５）最後までレンダリング。

【００３８】同様に、図８の例によれば、エッジ情報１５には、下記が含まれることになる。すなわち、

・エッジ８４は画素位置１００から始まり、エッジ８２は画素位置１００から始まる；

・エッジ９２は画素位置４０から始まり、エッジ９４は画素位置１６０から始まる；

・エッジ８４は７０スキャンライン延び、エッジ８２は

７０スキャンライン延びる；
・エッジ８４は傾斜＝－１を有し、エッジ８４は傾斜＝＋１を有する；
・エッジ９２は傾斜＝０を有し、エッジ９４は傾斜＝０を有する；
・エッジ９２および９４は７０スキャンラインだけ延びる。

【００３９】上の命令ストリーム１４およびエッジ情報１５の例とそれぞれが表現される形から、図８の画像７８では、画素位置（Ｘ）とスキャンライン値（Ｙ）によって、画像７８がレンダリングされる単一の出力空間が定義されることが認められる。しかし、本開示の原理を使用して、他の出力空間構成を実現することもできる。

【００４０】図８には、ラスタ画像画素データが含まれず、したがって、表示リスト１３の記憶部分１６には何も記憶する必要がない。この特徴については後述する。

【００４１】表示リスト格納部１３は、画素シーケンシャルレンダリング装置２０によって読み取られる。画素シーケンシャルレンダリング装置２０は、通常は集積回路を用いて実施される。画素シーケンシャルレンダリング装置２０は、表示リストをラスタ画素のストリームに変換し、このストリームは、たとえばプリンタ、ディスプレイまたはメモリ格納装置などの別の装置に転送され得る。

【００４２】なお、好ましい実施形態においては、画素シーケンシャルレンダリング装置２０を集積回路として説明するが、これは、ホストプロセッサ２などの汎用処理ユニット上で同等の処理を実行可能なソフトウェアモジュールによって実施することもできる。このソフトウェアモジュールは、ディスク装置またはコンピュータネットワークなどのコンピュータ可読媒体を介してユーザに配布することのできるコンピュータプログラム製品の一部を形成することができる。

【００４３】図３は、画素シーケンシャルレンダリング装置２０、表示リスト格納部１３および一時的なレンダリング格納部３０の構成を示す図である。画素シーケンシャルレンダリング装置２０の処理ステージ２２は、命令実行部３００（本明細書の「１．０　命令実行部」においてより詳細に説明する）、エッジ処理モジュール４００（本明細書の「２．０　エッジ処理モジュール」においてより詳細に説明する）、アクティビティ判断及び命令生成モジュール５００（本明細書の「３．０　アクティビティ判断及び命令生成モジュール」においてより詳細に説明する）、塗潰し色決定モジュール６００（本明細書の「４．０　塗潰し色決定モジュール」においてより詳細に説明する）、画素合成モジュール７００（本明細書の「５．０　画素合成モジュール」においてより詳細に説明する）、及び画素出力モジュール８００（本明細書の「６．０　画素出力モジュール」においてより詳細に説明する）を含む。

【００４４】また、本処理動作では一時的格納部３０を使用するが、これは、上で述べたように表示リスト格納部１３と同一の装置（たとえば磁気ディスクまたは半導体ＲＡＭ）を共用してもよいし、速度最適化のために個々に格納部（ストア）を用いて実施することができる。エッジ処理モジュール４００は、エッジレコード格納部３２を使用して、スキャンラインからスキャンラインへ順方向に運ばれるエッジ情報を保持する。アクティビティ判断及び命令生成モジュール５００は、レベルアクティベーションテーブル３４を使用して演算子性能に関する情報と、スキャンラインがレンダリングされている間のエッジ交差に関する各領域のその時点におけるアクティビティの状態に関する情報とを保持する。塗潰し色決定モジュール６００は、塗潰しデータテーブル３６を使用して、特定の位置で特定の優先順位の塗潰し色を決定するのに必要な情報を保持する。画素合成モジュール７００は、画素合成スタック３８を使用して、値を決定するために多数の優先順位からの色を要求する出力画素を決定する間、中間結果を保持する。

【００４５】表示リスト格納部１３および上記で詳細を示した他の格納部３２乃至３８は、ＲＡＭ内で実現するか、他のデータ記憶技術によって実現することができる。

【００４６】図３の実施形態に示された処理ステップは、処理パイプライン２２の形をとる。この場合、パイプラインのモジュールは、以下で説明する形態でそれらの間でメッセージを受け渡しながら、画像データの異なる部分に対して並列に同時に実行することができる。もう１つの実施形態では、以下で説明するメッセージのそれぞれが、下流モジュール制御への同期転送の形をとることができ、上流処理は、下流モジュールがそのメッセージの処理を完了するまで中断される。

【００４７】１．０　命令実行部
命令実行部３００は、命令ストリーム１４から命令を読み取り、処理し、その命令を、出力３９８を介してパイプライン２２内の他のモジュール４００、５００、６００および７００に転送されるメッセージにフォーマットする。好ましい実施形態では、命令ストリーム１４に、以下の命令を含めることができる。

【００４８】LOAD_PRIORITY_PROPERTIES：この命令は、レベルアクティベーションテーブル３４にロードされるデータと、そのデータがロードされるテーブル内のアドレスに関連する。命令実行部３００は、この命令を検出したとき、レベルアクティベーションテーブル３４の指定された位置にデータを格納するためのメッセージを発行する。これは、このデータを含むメッセージをフォーマットし、処理パイプライン２２を介して、格納動作を実行するアクティビティ判断及び命令生成モジュール５００に渡すことによって達成できる。

【００４９】LOAD_FILL_DATA：この命令は、塗潰しデー

タテーブル３６にロードされるデータと、そのデータがロードされるテーブル内のアドレスに関連する。命令実行部３００は、この命令を検出したとき、塗潰しデータテーブル３６の指定されたアドレスのデータを格納するためのメッセージを発行する。これは、このデータを含むメッセージをフォーマットし、処理パイプライン２２を介して、格納動作を実行する塗潰しデータ決定モジュールに渡すことによって達成できる。

【００５０】LOAD_NEW_EDGES_AND_RENDER：この命令は、次のスキャンラインをレンダリングする時にレンダリング処理に導入される新しいエッジ１５の表示リスト格納部１３内におけるアドレスに関連する。命令実行部３００は、この命令を検出すると、このデータを含むメッセージをフォーマットし、エッジ処理モジュール４００に渡す。エッジ処理モジュール４００は、新しいエッジのアドレスをエッジレコード格納部３２に格納する。指定されたアドレスにあるエッジは、次のスキャンラインをレンダリングする前に、最初のスキャンライン交差座標に基づいてソートされる。一つの実施形態では、エッジは、表示リスト生成処理１２によってソートされる。また、別の実施形態では、エッジは、画素シーケンシャルレンダリング装置２０によってソートされる。

【００５１】SET_SCAN LINE_LENGTH：この命令は、レンダリングされるスキャンラインのそれぞれで作られる画素数に関連する。命令実行部３００は、この命令を検出すると、この値をエッジ処理モジュール４００および画素合成モジュール７００に渡す。

【００５２】SET_OPACITY_MODE：この命令は、画素合成演算で不透明度チャネル（当技術分野ではアルファチャネルとも称する）を使用するかどうかを示すフラグに関連する。命令実行部３００は、この命令を検出すると、このフラグの値を画素合成モジュール７００に渡す。

【００５３】命令実行部３００は、通常は、命令をマッピングし、パイプライン動作に復号して、さまざまなモジュールに渡す、マイクロコードステートマシンによって形成されるが、その代わりに、対応するソフトウェア処理を使用することもできる。

【００５４】２．０　エッジ処理モジュール

スキャンラインのレンダリング動作中のエッジ処理モジュール４００の動作を、図４を参照して以下に説明する。スキャンラインのレンダリングのための初期条件は、以下の３つのエッジレコードのリストが使用可能であることである。これら３つのリストのいずれかまたはすべては空でもよい。これらのリストは、エッジ情報１５から取得されLOAD_NEW_EDGES_AND_RENDER命令によってセットされる、新しいエッジを含む新エッジリスト４０２、前のスキャンラインから順方向に運ばれたエッジレコードを含む主エッジリスト４０４および、やはり前のスキャンラインから順方向に運ばれたエッジレコードを含むスピルエッジリスト４０６である。各エッジレコードには、下記が含まれ得る。

【００５５】（ｉ）現時点のスキャンラインの交差の座標（本明細書ではＸ座標と称する）、（ii）現時点のこのエッジの線分が続くスキャンラインの数（本明細書ではNYと称する。また、いくつかの実施形態では、これをＹ限界と表現する場合がある）、（iii）各スキャンラインの後でこのエッジレコードのＸ座標に加算される値（本明細書ではDXと称する）、（iv）各スキャンラインの後でこのエッジレコードのDXに加算される値（本明細書ではDDXと称する）、（ｖ）１つまたは複数の優先順位番号（Ｐ）、（vi）エッジがスキャンラインと上向き（＋）に交差するか下向き（－）に交差するかを示す方向フラグ（DIR）、（vii）リスト内の次のエッジ線分のアドレス（ADD）。

【００５６】このようなフォーマットは、ベクトル、直交配置されたエッジおよび二次曲線に適合する。これ以外のパラメータ、たとえばDDDXの追加によって、このような配置が三次曲線に適合できるようになる。三次ベジェスプラインなど、いくつかの応用分野では、６階多項式（すなわちDDDDDDXまで）が必要になる場合がある。

【００５７】図８のエッジ８４および９４の例では、スキャンライン２０での対応するエッジレコードが、図１６において示されるテーブルのようになる。

【００５８】この説明では、レンダリング処理によって生成されつつあるスキャンラインに沿って画素から画素へステップする座標を、Ｘ座標と称し、スキャンラインからスキャンラインへとステップする座標を、Ｙ座標と称する。各エッジリストには、メモリ内で連続的に配置された０個以上のレコードが含まれることが好ましい。ポインタチェインの使用を含む他の記憶配置も可能である。３つのリスト４０２、４０４および４０６のそれぞれのレコードは、スキャンライン交差（Ｘ）座標の順で配置される。これは、通常は、最初は、エッジ情報を含むメッセージを命令実行部３００から受け取るエッジ入力モジュール４０８によって管理されるソート処理によって得られる。各スキャンライン交差座標の整数部分だけを有意とみなすためにソートを簡易化することが可能である。また、各スキャンラインの交差座標を、現在のレンダリング処理によって作られる最小および最大のＸ座標にクランプされるとみなすことによってさらにソートを簡易化することが可能である。適切な場合には、エッジ入力モジュール４０８は、メッセージを、出力４９８を介してパイプライン２２の下流のモジュール５００、６００および７００に受け渡す。

【００５９】エッジ入力モジュール４０８は、３つのリスト４０２、４０４および４０６のそれぞれへの参照を継続し、これらのリストからエッジデータを受け取る。これらの参照のそれぞれは、スキャンラインの処理の開始時に、各リスト内の最初のエッジを参照するように初期設定される。その後、エッジ入力モジュール４０８

は、3つの参照されるレコードのうちの最小のX座標を有するレコードを選択するべく、3つの参照されたエッジレコードのうちの1つを選択する。複数のレコードのXが等しい場合には、それぞれが任意の順序で処理され、対応するエッジ交差が、下記の形で出力される。そのレコードの選択に使用された参照は、その後、そのリストの次のレコードに進められる。選択されたばかりのエッジは、メッセージにフォーマットされ、エッジ更新モジュール４１０に送られる。また、エッジのいくつかのフラグ、具体的には現在のX、優先順位番号および方向フラグが、メッセージにフォーマットされ、このメッセージは、エッジ処理モジュール４００の出力４９８としてアクティビティ判断及び命令生成モジュール５００に転送される。なお、本明細書に記載されたものより多数または少数のリストを使用する実施形態も可能である。

【００６０】エッジを受け取った時に、エッジ更新モジュール４１０は、現在の線分が続くスキャンライン数のカウントをデクリメントする。そのカウントが0に達した場合には、次の線分アドレスによって示されるアドレスから新しい線分を読み取る。線分では、下記が指定される。

（ｉ）線分を読み取った直後に現在のX座標に加算される値、（ii）そのエッジの新しいDX値、（iii）そのエッジの新しいDDX値、（iv）新しい線分が続くスキャンライン数の新しいカウント。

【００６１】示されたアドレスに使用可能な次の線分がない場合には、そのエッジに対してそれ以上の処理は実行されない。そうでない場合には、エッジ更新モジュール４１０は、そのエッジの次のスキャンラインにおけるX座標を計算する。これには、通常は、現在のX座標をとり、これにDX値を加算することが用いられる。DXは、処理されるエッジの種類に応じて、必要であればDDX値を加算される場合がある。その後、エッジは、複数のエッジレコードの配列であるエッジプール４１２内の使用可能な空きスロットに書き込まれる。空きスロットがない場合には、エッジ更新モジュール４１０は、スロットが使用可能になるのを待つ。エッジレコードがエッジプール４１２に書き込まれたならば、エッジ更新モジュール４１０は、新しいエッジがエッジプール４１２に追加されたことを、信号線４１６を介してエッジ出力モジュール４１４に知らせる。

【００６２】スキャンラインのレンダリングのための初期条件として、エッジ出力モジュール４１４は、図4には図示されていないが、エッジレコード３２内のリスト４０４および４０６に関連する次の主エッジリスト４２０および次のスピルエッジリスト４２２のそれぞれへの参照を有する。これらの参照のそれぞれは、当初は空のリスト４２０および４２２が構築される位置に初期設定される。エッジ出力モジュール４１４は、エッジがエッジプール４１２に追加されたことを示す信号４１６を受け取ったとき、追加されたエッジが、次の主エッジリスト４２０に最後に書き込まれたエッジ（があれば）より小さいX座標を有するかどうかを判定する。これが真である場合には、順序付けの基準を侵害せずにそのエッジを主エッジリスト４０４の末尾に追加することができないので、「スピル」が発生したという。スピルが発生したときには、そのエッジは、好ましくはソートされた次のスピルエッジリスト４２２を維持する形で、次のスピルエッジリスト４２２に挿入される。たとえば、これは、ソフトウェアソートルーチンを使用して達成することができる。いくつかの実施形態では、スピルが、極端に大きいX座標など、他の条件によってトリガされる場合がある。

【００６３】エッジプール４１２に追加されたエッジが、次の主エッジリスト４２０に最後に書き込まれたエッジ（があれば）以上のX座標を有し、エッジプール４１２に使用可能な空きスロットがない場合には、エッジ出力モジュール４１４は、エッジプール４１２から、最小のX座標を有するエッジを選択し、そのエッジを次の主エッジリスト４２０の末尾に追加し、この処理で次の主エッジリスト４２０を延長する。エッジプール４１２内の、そのエッジによって占められていたスロットは、空きとしてマークされる。

【００６４】エッジ入力モジュール４０８は、これら3つの入力リスト４０２、４０４および４０６のすべてからすべてのエッジを読み取り、転送した後に、スキャンラインの終りに達したことを示すメッセージを形成し、そのメッセージを、アクティビティ判断及び命令生成モジュール５００とエッジ更新モジュール４１０の両方に送る。エッジ更新モジュール４１０は、そのメッセージを受け取ると、現在実行しているすべての処理が完了するのを待ち、その後、このメッセージをエッジ出力モジュール４１４に転送する。そのメッセージを受け取ったエッジ出力モジュール４１４は、エッジプール４１２からの残りのエッジレコードのすべてを、X順で次の主エッジリスト４０４に書き込む。その後、次の主エッジリスト４２０および主エッジリスト４０４への参照を、エッジ入力モジュール４０８とエッジ出力モジュール４１４の間で交換し、同様の交換を、次のスピルエッジリスト４２２とスピルエッジリスト４０６に関しても実行する。この形で、次のスキャンラインのための初期条件が確立される。

【００６５】エッジレコードの挿入時に次スピルエッジリスト４２２をソートするのではなく、そのようなエッジレコードを、単にリスト４２２の末尾に追加することができ、スキャンラインの末尾であって現在のスピルリスト４０６との交換の前にソートされたリスト４２２は、次のスキャンラインのエッジラスタ化でアクティブになる。エッジをソートする他の方法において、より少

数またはより多数のリストを使用することができ、また、異なるソートアルゴリズムを使用することができる。

【００６６】上記から、エッジ交差メッセージが、スキャンライン順および画素順（すなわち、まずＹでソートされ、次にＸでソートされる）でアクティビティ判断及び命令生成モジュール５００に送られること、および各エッジ交差メッセージに、それに適用される優先順位のラベルが付けられることを推論できる。

【００６７】図１２Ａは、エッジの線分が受け取られる時にエッジ処理モジュール４００によって作成され得るアクティブエッジレコード４１８の具体的な構造を示す図である。エッジの最初の線分がステップ（直交）線分である場合には、エッジのＸ値を、最初の線分の「Ｘステップ」と称する変数に加算して、アクティブ化されたエッジのＸ位置を得る。そうでない場合には、エッジのＸ値を使用する。これは、新しいエッジレコードのエッジが、$X_{edge}+X_{step}$によってソートされなければならないことを意味する。したがって、最初の線分のX_{step}は、エッジのソートを単純化するために０でなければならない。最初の線分のＹ値は、アクティブエッジレコード４１８のＮＹフィールドにロードされる。アクティブなエッジのＤＸフィールドは、ベクトルまたは二次線分のＤＸフィールド識別子からコピーされ、ステップ線分の場合には０がセットされる。図１２Ａに示されたｕフラグは、線分が上向き（図１３Ａに関連する説明を参照されたい）である場合にセットされる。ｑフラグは、線分が二次線分の場合にセットされ、そうでない場合にはクリアされる。ｉフラグが設けられ、これは、線分が不可視である場合にセットされる。ｄフラグは、エッジが、関連するクリッピングレベルのない直接的なクリッピングオブジェクトして使用され、閉じた曲線に適用可能である時にセットされる。線分の実際の優先順位レベルまたはレベルアドレスは、新しいエッジレコードの対応するフィールドからアクティブエッジレコード４１８のレベル（ADDR）フィールド（Level(Addr)）にコピーされる。アクティブエッジレコード４１８の線分アドレス／ＤＤＸフィールド（Seg.Addr(DDX)）は、線分リスト内の次の線分のアドレスであるか、線分が二次線分の場合にセグメントのＤＤＸ値からコピーされるかのいずれかである。線分アドレスは、エッジレコードを終了させるのに使用される。その結果、好ましい実施形態では、全ての二次曲線（すなわち、ＤＤＸフィールドを使用する曲線）が、エッジレコードの終端線分になる。

【００６８】図１２Ａから、他のデータ構造も可能であり、たとえばより高次の多項式の実装が使用される場合にはそれが必要であることを諒解されたい。さらに、線分アドレスおよびＤＤＸフィールドは、異なるフィールドに分離することができ、代替実施態様に合わせて追加のフラグを設けることができる。

【００６９】図１２Ｂは、エッジ処理モジュール４００で使用される、上で説明した好ましい実施形態のエッジレコードの配置を示す図である。エッジプール４１２は、新アクティブエッジレコード４２８、現アクティブエッジレコード４３０およびスピルアクティブエッジレコード４３２によって補足される。図１２Ｂからわかるように、レコード４０２、４０４、４０６、４２０および４２２は、ある時点でレンダリングされるエッジの数に応じてそのサイズを動的に変更することができる。各レコードには、新エッジリスト４０２の場合にはLOAD_EDGES_AND_RENDER命令に組み込まれたSIZE値によって決定される、限界値が含まれる。このような命令が検出された場合には、SIZEを検査し、０でない場合には、新しいエッジレコードのアドレスをロードし、リスト４０２の限界サイズを決定する限界値を計算する。

【００７０】好ましい実施形態では、エッジレコードの処理のために配列とそれに関連するポインタを使用するが、たとえばリンクリストなどの、他の実施態様を使用することができる。これらの他の実施態様は、ハードウェアベース、ソフトウェアベースまたはその組合せとすることができる。

【００７１】図８に示された画像７８の具体的なレンダリングを、図１０に示されたスキャンライン３４、３５および３６に関連して以下に説明する。この例では、次のスキャンラインの新しいＸ座標の計算が、説明を明瞭にするために省略され、図１２Ｃないし図１２Ｉでは、出力エッジ交差が、エッジプール４１２のレジスタ４２８、４３０および４３２の１つから導出される。

【００７２】図１２Ｃは、スキャンライン３４（半透明の青い三角形８０の最上部）のレンダリングの終りでの、上で述べたリストの状態を示す図である。スキャンライン３４には、新しいエッジがなく、したがって、リスト４０２が空であることに留意されたい。主エッジリスト４０４および次主エッジリスト４２０のそれぞれには、エッジ８２および８４だけが含まれる。リストのそれぞれには、対応するポインタ４３４、４３６および４４０が含まれ、これらは、スキャンライン３４の完了時に、対応するリスト内の次の空きレコードを指す。各リストには、対応するリストの末尾を指すために必要な、アスタリスク（＊）によって示される限界ポインタ４５０も含まれる。リンクリストを使用する場合には、リンクリストに、対応する機能を実行するヌルポインタ終端子が含まれるので、このような限界ポインタは不要になる。

【００７３】上で述べたように、各スキャンラインの始めに、次の主エッジリスト４２０と主エッジリスト４０４が交換され、新しいエッジが、新エッジリスト４０２に受け取られる。残りのリストはクリアされ、ポインタのそれぞれは、各リストの最初のメンバを指すようにセットされる。スキャンライン３５の始めでは、配置は図

１２Ｄのようになる。図１２Ｄからわかるように、レコードには、4つのアクティブなエッジが含まれ、図１０からわかるように、エッジ９２、９４、８４および８２に対応する。

【００７４】図１２Ｅを参照すると、レンダリングが開始されるときに、新エッジレコード４０２の最初の線分が、アクティブエッジレコード４２８にロードされ、主エッジリスト４０４およびスピルエッジリスト４０６の最初のアクティブなエッジレコードが、それぞれレコード４３０および４３２にコピーされる。この例では、スピルエッジリスト４０６は空であり、したがって、ローディングは行われない。レコード４２８、４３０および４３２内のエッジのＸ位置が比較され、エッジ交差が、最小のＸ位置を有するエッジについて発行される。この場合、発行されるエッジは、エッジ９２に対応するエッジであり、このエッジがその優先順位値と共に出力される。その後、ポインタ４３４、４３６および４３８が、リスト内の次のレコードを指すように更新される。

【００７５】その後、エッジ交差が発行されるエッジが更新され（この場合、その位置にDX＝０を加算することによって）、エッジプール４１２にバッファリングされる。このエッジプール４１２は、この例では3つのエッジレコードを保持するサイズになる。発行されたエッジが現れたリスト（この場合ではリスト４０２）内の次の項目が、対応するレコード（この場合ではレコード４２８）にロードされる。これを図１２Ｆに示す。

【００７６】さらに、図１２Ｆから明らかなとおり、レジスタ４２８、４３０および４３２の間の比較によって、もう一度最小のＸ値を有するエッジが選択され、適切な次のエッジ交差（Ｘ＝８５、Ｐ＝２）として出力される。そして、同様に、選択され出力されたエッジは、更新され、エッジプール４１２に追加され、適当なポインタのすべてがインクリメントされる。この場合、更新される値は、Ｘ←Ｘ＋DXによって与えられ、ここでは、８４＝８５－１として与えられる。また、図からわかるように、新しいエッジのポインタ４３４が、この場合では新エッジリスト４０２の末尾に移動される。

【００７７】図１２Ｇでは、現在の最小のＸ値を有すると識別された次のエッジが、やはり、レジスタ４３０から取得され、エッジ交差（Ｘ＝１１５、Ｐ＝２）として出力される。そして、エッジの更新が発生し、図示のように、その値がエッジプール４１２に追加される。この時、エッジプール４１２は満杯になり、ここから、最小のＸ値を有するエッジが選択され、出力リスト４２０に発行され、対応する限界ポインタが、それに相応して移動される。

【００７８】図１２Ｈからわかるように、次の最小のエッジ交差は、レジスタ４２８からのものであり、これが出力される（Ｘ＝１６０、Ｐ＝１）。やはりエッジプール４１２が更新され、次に小さいＸ値が出力リスト４２０に発行される。

【００７９】スキャンライン３５の終りに、図１２Ｉからわかるように、Ｘ値の小さい順で、エッジプール４１２の内容が出力リスト４２０にフラッシュされる。図１２Ｊからわかるように、次の主エッジリスト４２０と主エッジリスト４０４は、次のスキャンライン３６のレンダリングに備えて、ポインタを交換することによって交換される。交換の後に、図１２Ｊからわかるように、主エッジリスト４０４の内容には、スキャンライン３６上の現在のエッジのすべてが含まれ、これらはＸ位置の順で配置され、これによって、高速のレンダリングを容易にする便利なアクセスが可能になる。

【００８０】通常、新しいエッジは、Ｘ位置の昇順でエッジ処理モジュール４００によって受け取られる。新しいエッジが現れる時には、その位置が更新され（次にレンダリングされるスキャンラインのために計算され）、これによって、以下の処置が決定される。

（ａ）更新された位置が信号線４９８に出力された最後のＸ位置より小さい場合には、新しいエッジは、主スピルリスト４０６へのソートされる挿入であり、対応する限界レジスタが更新される。

（ｂ）そうでない場合に、空間があるならば、そのエッジはエッジプール４１２内で保存される。

【００８１】前述から明白なとおり、エッジプール４１２は、ラスタ化画像の次のスキャンラインのレンダリングに備えた順序付きの形でのリストの更新を補助する。さらに、エッジプール４１２のサイズは、多数の順序付けられていないエッジに適合するために変更することができる。しかし、実際には、エッジプール４１２が、一般にグラフィック処理システムの処理速度および使用可能なメモリに依存する、実用的な限界を有することは明白である。制限的な意味では、エッジプール４１２を省略することができるが、これは、通常は、更新されたエッジを、次の出力エッジリスト４２０へのソートされた挿入にすることを必要とする。しかし、好ましい実施形態では、上で述べたスピルリストの使用を介することにより、必然的にこの状況が回避される。スピルリストを設けることによって、好ましい実施形態を、実用的なサイズのエッジプールを用いて実施でき、なおかつ、ソフトウェア集中的なソート手順に頼らずに比較的複雑なエッジ交差を処理することができる。稀ではあるが、エッジプールとスピルリストがエッジ交差の複雑さに適合するのに不十分になった場合では、ソート法を使用するようにしてもよい。

【００８２】スピルリスト手順が使用される場合の例を、図１３Ａに示す。図１３Ａでは、３つの任意のエッジ６０、６１および６３が、スキャンラインＡおよびＢの間の相対的位置で任意のエッジ６２と交差する。さらに、スキャンラインＡおよびＢのそれぞれについて実際に表示される画素位置６４が、スパン画素位置Ｃ乃至Ｊ

として図示されている。エッジプール４１２が3つのエッジレコードを保存するサイズであるとした上記の例では、このような配置だけでは、図１３Ａに示された隣接するスキャンラインの間で発生する3つのエッジの交差に適合するのに不十分であることは明白である。

【００８３】図１３Ｂに、スキャンライン上のエッジ６０、６１および６３をレンダリングした後のエッジレコードの状態を示す。エッジ交差Ｈは、最も最近に発行されたエッジ交差であり、エッジプール４１２は、次のスキャンラインであるスキャンラインＢのための、それぞれエッジ６０、６１および６３の更新されたＸ値Ｅ、ＧおよびＩで満杯になっている。エッジ６２は、現アクティブエッジレコード４３０にロードされ、エッジプール４１２が満杯なので、エッジ６０に対応する最小のＸ値が、出力エッジリスト４２０に出力される。

【００８４】図１３Ｃでは、次のエッジ交差が発行され（エッジ６２、Ｘ＝Ｊ）、対応する更新された値、この場合はスキャンラインＢのＸ＝Ｃが決定される。新たに更新された値Ｘ＝Ｃは、出力リスト４２０からコピーされた最も最近の値Ｘ＝Ｅより小さいので、現在のエッジレコードとそれに対応する新たに更新された値が、出力スピルリスト４２２に直接に転送される。

【００８５】図１３Ｄに、スキャンラインＢの開始時のエッジレコードの状態を示す。この図では、主リストおよび出力リストと、それらに対応するスピル構成要素が交換されていることがわかる。最初に発行されるエッジを決定するために、エッジ６０を現アクティブエッジレジスタ４３０にロードし、エッジ６２を、スピルアクティブエッジレジスタ４３２にロードする。Ｘ値が比較され、最小のＸ値（Ｘ＝Ｃ）を有するエッジ６２が発行され、更新され、エッジプール４１２にロードされる。

【００８６】エッジの発行と更新は、主エッジリスト４０４内の残りのエッジについて継続され、スキャンラインの終りに、エッジプール４１２がフラッシュされて、図１３Ｅに示された状況になる。図１３Ｅからは、エッジ６０ないし６３のそれぞれが、次のスキャンラインでのレンダリングのために適当に順序付けられ、スキャンラインＢ上で正しく発行され、レンダリングされたことがわかる。

【００８７】上記から明らかになるように、スピルリストは、複雑なエッジ交差状況の存在のもとで、エッジラスタ化順序の維持をもたらす。さらに、このリストがサイズにおいて動的に変更可能であることによって、エッジ交差の数と複雑さの大きい変化を、例外的に複雑なエッジ交差以外のすべてのエッジ交差でソート手順に頼る必要なしに処理できる。

【００８８】好ましい実施形態では、エッジプール４１２は、8つのエッジレコードを保存するサイズにされ、リスト４０４および４２０のサイズは、それらに関連するスピルリスト４０６および４２２と一緒に、動的に変更可能であり、これによって、複雑なエッジ交差要件を有する大きい画像を処理するために十分な範囲が提供される。

【００８９】３．０　アクティビティ判断及び命令生成モジュール

表現ツリーはイメージを形成するのに必要となる合成演算を記述するのにしばしば使用され、通常は、リーフノード、単項ノードおよびバイナリノードを含む多くのノードを備える。リーフノードは表現ツリーの一番はずれのノードであり、これに続く下位のノードを持たず、イメージのプリミティブな構成要素、すなわちグラフィックオブジェクトを表す。単項ノードは、単項演算子の下位のツリーの部分より生じるオブジェクトの画素データを変更する操作を表す。通常、バイナリのノードは左右のサブツリーに分岐する。ここで、サブツリーはそれぞれ少なくとも1つのリーフノードを備える。バイナリのノードは1つのブランチ上の1つのオブジェクトと、他のブランチ上の他のオブジェクトとの間の演算を表す。

【００９０】そのような演算に関する例は「ディジタルイメージの合成（Compositing Digital Images）」（Porter、T及びDuff、T、コンピュータグラフィックス、Vol.18No.3(1984)、253-259ページ）において記述されている。これらのPorterとDuffの合成演算のいくつかを図１５に示す。図示を容易にするために、図１５で示されるグラフィックオブジェクトは完全に不透明であるとする。図からわかるように、これらの演算子のいくつか（例えば、out）はデータ（例えば、色データ）を、２個のオブジェクトの中央のオーバラッピング領域に提供しない。一般に、２個のオブジェクトの1つもしくは両方が部分的に透明であるならば、これらの演算子（例えば、out）は常にデータ（例えば、色データ）をそれら２個のオブジェクトの中央オーバラッピング領域に提供することになる。しかしながら、２個のオブジェクトの1つもしくは両方が完全に不透明であるならば、特定の演算子がいかなるデータをもこれら２個のオブジェクトの中央オーバラッピング領域に提供しないという特別な場合が生じることになる。

【００９１】次に、図１９をみると、単純な表現ツリーとこれに対応する命令リストの典型的な例が示されている。表現ツリーは、それぞれグラフィックオブジェクトＡ、Ｂ、Ｃ、ＤおよびPAGEを記述するリーフノード１０、９、７、６及び０－５を備える。また、表現ツリーは、2つのブランチを有し、それぞれ演算、“out”、“in”、“over”および“over”を表わすノード８、１１、１２、および１３を含む。命令リストは表現ツリーを描画するための命令のリストを含んでいる。しかしながら、合成演算が、その演算によって合成されるグラフィックオブジェクトのどれが与えられた画素位置で有効であるかということに影響されるので、与えられた画素位置に対して命令リストを生成するときには問題が生じ

る。好適なアクティビティ判断及び命令生成モジュール５００は、これらの問題を解決しようとするものである。

【００９２】次に、アクティビティ判断及び命令生成モジュール５００の動作について図５を参照して説明する。初期化処理段階の間、命令生成部３００は、レンダリングされるイメージのオブジェクト図形記述をレベルアクティベーションテーブル生成部５０２に渡す。このオブジェクト図形記述は、例えば図１９で示される表現ツリーのような、表現ツリーの形態をとる。次に、レベルアクティベーションテーブル生成部５０２は、レベルアクティベーションテーブルを生成する。このテーブルは、表現ツリーの１次形式であり、レベルアクティベーションテーブル格納部３４に格納される。生成されたレベルアクティベーションテーブルは複数のレコードを含んでおり、表現ツリーの２項演算子ノード、単項ノード、及びリーフノードの各々に対して一つのノードが割り当てられている。

【００９３】さらに、また、レベルアクティベーションテーブルは２項演算子のある特有な性質に関するデータ格納のためのフィールドと、２項演算子ノードのブランチのアクティビティに関するデータ格納のためのフィールドとを含んでいる。２項演算子の性質に関するデータは静的である。すなわち、現在スキャンされた画素の位置に依存するのではなく、使用される実際の演算子に依存する。その結果、初期化処理段階の間に、このデータは生成され、レベルアクティベーションテーブルの中に格納されることができる。しかしながら、ブランチのアクティビティに関するデータは、現在スキャンされている画素位置に依存する。初期化処理段階の間は、その時点で有効な画素位置はないので、レベルアクティベーションテーブルのブランチアクティビティデータは無効な状態に初期化される。

【００９４】そして、画素順次描画装置２０は、ラスタスキャン順次で、イメージの画素位置をスキャンし始める。エッジ処理モジュール４００はレベルアクティベーションテーブル更新モジュール５０４に、スキャンされている現在のスキャンラインのエッジの画素位置を渡す。レベルアクティベーションテーブル更新モジュール５０４は格納部３４からレベルアクティベーションテーブルにアクセスして、どの画素位置が現在スキャンされているのかに依存してテーブルを更新する。実現の容易化のために、テーブルは隣接しているエッジの間のそれらの画素位置に対して一度だけアップデートされる。これは、スキャンラインの２つの隣接するエッジの間の画素位置の１つのグループ内では、表現ツリーをレンダリングするための命令リストは同一であるということの認識に基づいている。或いは、それぞれのスキャンされた画素位置に対してテーブルはアップデートされる構成とすることもできる。

【００９５】トラバーサル及び命令生成モジュール５０６は、次に、この更新されたレベルアクティベーションテーブルに基づいて命令を生成する。その命令は、次に、塗潰し色決定モジュール６００に渡される。イメージが描画されるまで、隣接しているエッジの間の画素位置の各グループに対して、このプロセスは持続される。

【００９６】本実施形態の更なる説明に先立って、本明細書において使用される用語について簡単に説明する。

【００９７】図１８Ａと１８Ｂをみると、ソースオブジェクトＬと目的オブジェクトＲの間の２進演算（表現ツリーとして表されている）が示される。表現ツリーには、オブジェクトＬへの左のブランチとオブジェクトＲへの右のブランチを有するバイナリノードＯＰが示される。オブジェクトに適用される境界エッジの中に現在スキャンされている画素があるならば、オブジェクトはその画素にアクティブであるとみなされる。境界エッジの外に画素があるならば、オブジェクトはその画素に非アクティブである。こうして、境界エッジの中のそれらの画素のグループを走査する間、そのオブジェクトはアクティブになる。オブジェクトがアクティブである場合、オブジェクトへの演算子のブランチはアクティブであるとみなされる。実際に実行される動作とは関係なく、図１８Ａの２進演算は以下で示されるような、４つのアクティビティの領域に分解される。すなわち、

領域１：Ｌオブジェクトがアクティブでオブジェクトが無効(L∩/R)、

領域２：Ｌオブジェクトがアクティブでオブジェクトもアクティブ(L∩R)、

領域３：Ｌオブジェクトが無効でＲオブジェクトがアクティブ(/L∩R)、

領域４：Ｌオブジェクトが無効でＲオブジェクトも無効（／(L∪R)）。

【００９８】ここで、領域４は、常に、無演算（ＮＯＰ）の実行が要求されることになる。従って、結果として、バイナリツリーのための有効なレベルとしては３つの異なった組合せが存在することになる。説明の簡単化のために、ここでは、領域(L∩/R)を左のノード領域或いは左の領域と称し、領域(/L∩R)を右のノード領域或いは右の領域と称し、領域(L∩R)を共通の領域と称する。左のノード領域と右ノード領域は、それぞれ親ノードの左及び右ブランチと関連し、イメージにおけるそれらの位置に対応するものではなく、オブジェクト演算子の順序（たとえば、Ｌｏｐ Ｒといった順序）に対応している。

【００９９】この説明の目的のために、２項演算子がデータをツリーに提供するならば、その２項演算子はアクティブであるとみなされる。２項演算子のアクティビティは演算子自体の特有な性質とその左右のブランチのアクティビティである。

【０１００】例として、図１８Ａと図１５の“out”演

算子を比べてみる。“out”演算子には、領域(L∩/R)に関して、先頭のオブジェクト（左ブランチ）が部分的に透明であるか不透明であるかの如何にかかわらずツリーにデータを渡し、そして、共通の領域(L∩R)では、後続のオブジェクト（右のブランチ）が部分的に透明である場合にのみ、ツリーにデータを渡すという特有な性質がある。前者の場合は、現在スキャンされている画素のグループに対して、左ブランチがアクティブに、右ブランチが非アクティブになる場合にのみ発生する。後者の場合は、左右ブランチが、現在スキャンされた画素のグループに対してともにアクティブであるときに発生する。他方、“out”演算子には、後続（右ブランチ）のオブジェクトが部分的に透明であるかまたは不透明であるかにかかわらず、領域(/L∩R)に対して、データをツリーに渡さないという特有な性質がある。

【０１０１】別の例として、図１８Ａと図１５の“in”演算子を比べてみる。“in”演算子には、領域(L∩R)に関して、データ（例えば色データ）をツリーに渡す特有な性質がある。しかしながら、これは、その左右ブランチがアクティブであるときにだけ生じる。他方、“in”演算子には、領域(/L∩R)と(L∩/R)に対し、どんなデータもツリーに渡さないという特有な性質がある。

【０１０２】別の例として、図１８Ａと図１５の“ROUT”演算子を比較する。“ROUT”演算子には、領域(/L∩R)に関して、先頭の（右ブランチ）オブジェクトが部分的に透明であるか不透明であるかにかかわらずデータをツリーに渡し、共通領域(L∩R)では、後続の（左ブランチ）オブジェクトが部分的に透明である場合にのみデータをツリーに渡す特有の性質がある。前者の場合は、現在スキャンされた画素のグループに対してその左ブランチが非アクティブになり、右のブランチがアクティブになる場合にのみ発生する。後者の場合は、左右ブランチが現在スキャンされた画素のグループに対して共にアクティブであるときにだけ発生する。他方、“ROUT”演算子には、領域(L∩/R)に関して、下側（左ブランチ）のオブジェクトが部分的に透明であるかまたは不透明であるかにかかわらず、いかなるデータもツリーに渡さないという特有な性質がある。

【０１０３】２項演算子のパフォーマンスとそのアクティビティの間には区別があることに注意すべきである。その両方のブランチがアクティブである場合にのみ、２項演算子は演算動作を実行する。さもなければ、演算子が要求する場合に、その結果は直接にアクティブなブランチから取得される。

【０１０４】アクティビティ判断及び命令生成モジュール５００は、２項演算子のアクティビティと、表現ツリーのブランチのアクティビティ、およびグラフィックオブジェクト（リーフノード）のアクティビティに基づいてツリーをなぞり、命令を生成する。

【０１０５】これらの命令が生成される方法について、図２０で示される例を参照して説明する。図２０は、図１９の表現ツリーにおいてオブジェクトＣがアクティブである状態を示すとともに、アクティビティ判断及び命令生成モジュール５００によって生成された、対応する命令リストを示す。モジュール５００は、まず最初に、表現ツリーの演算子の前述した特有の性質を判断する。これらの性質は、モジュール５００によって、検索と利用のために永久的（不揮発）に格納されている（不図示）。図２０において示される例では、モジュール５００は、記憶機構から“over”、“in”、および“out”演算子の２項演算子の性質を検索する。これらの性質は論理形式で格納されている。例えば、“over”演算子の性質は(/L∩R)=TRUEかつ(L∩/R)=TRUEとして格納される。すなわち、“over”演算子には、これらの領域に関してデータをツリーに渡す能力がある。別の例では、“in”演算子の性質は(/L∩R)=FALSEかつ(L∩/R)=FALSEとして格納される。すなわち、“in”演算子には、これらの領域に関して、ツリーにデータを渡す能力がない。更に別の例では、“out”演算子の性質は(/L∩R)=TRUEかつ(L∩/R)=FALSEとして格納される。以上の説明より明らかなように、領域(L∩R)の２項演算子の特有の性質はモジュール５００のパフォーマンスに対して不要であり、メモリに格納されない。そのノードの２進演算は、左右ブランチがアクティブであるときにいつも実行されるからである。

【０１０６】図２０の例において、初期化処理段階の後、アクティビティ判断及び生成モジュール５００は、スキャンされた画素位置の特定のグループのために、そのオブジェクトＣがアクティブで、オブジェクトＡ、Ｂ、およびＤが非アクティブであることを判断する。モジュール５００はオブジェクトのアクティビティから、それぞれのバイナリノードのブランチのアクティビティを決定する。すなわち、アクティブなリーフノードオブジェクト（例えばオブジェクトＣ及びPAGE）をルートノード（例えば、１３）に直接に結合させている全てのブランチがアクティブとなる。この説明において、アクティブな左ブランチはL Active=TRUEとして示され、非アクティブな左ブランチはL Active=FALSEとして示され、アクティブな右ブランチはR Active=TRUEとして示され、非アクティブな右ブランチはR Active=FALSEとして示される。図２０の例において、モジュール５００は以下のことを決定する。

【０１０７】（ｉ）バイナリノード１３の左右ブランチはアクティブである（すなわち、LActive=TRUE、かつR Active=TRUE）、（ii）バイナリノード１２で 左ブランチは非アクティブであり、右ブランチはアクティブである（すなわち、L Active=FALSE、かつR Active=TRUE）、（iii）バイナリノード１１の左右ブランチは非アクティブである（すなわち、L Active=FALSE、かつR Active=FALSE)、（iv）バイナリノード８で、左ブランチ

はアクティブであり、右ブランチは非アクティブである（すなわち、L Active=TRUE、かつR Active=FALSE）。

【0108】この説明の目的のために、本明細書ではシンボル&&、||、及び！を、それぞれバイナリ論理演算子のAND、OR、NOTとして用いる。

【0109】次に、アクティビティ判断及び命令生成モジュール500は、現在スキャンされた画素位置のグループについてなぞり、命令を生成する。それは2項演算子のアクティビティ、表現ツリーのブランチのアクティビティ、およびグラフィックオブジェクト（リーフノード）のアクティビティに基づいている。モジュール500は、上から下へ、左から右という方法で、ルートノード（例えば、ノード13）で始まる表現ツリーをなぞる。

【0110】ここで、モジュール500は、以下の条件を満たす表現ツリーのブランチのみをなぞる。すなわち、

（1）先になぞられたバイナリノードに関して、（L active && R active）||（(L∩/R)&& L active && !R active）==TRUEである場合は、先になぞられたバイナリノードの左ブランチの下方ノードへなぞる（トラバースする）。

（2）先になぞられたバイナリノードに関して、（L active && R active）||(/L∩R)&& !L active && R active）=TRUEの場合、先になぞられたバイナリノードの右ブランチの下方ノードへなぞる。

【0111】このなぞり（トラバース）と同時間の間、アクティビティ判断及び命令生成モジュール500は、アクティブな左右ブランチを有するなぞられたバイナリノードのための演算子命令と、なぞられたあらゆるリーフノードのためのリーフ値命令を生成する。

【0112】図20の例において、アクティビティ判断及び命令生成モジュール500は、ルートノード13からそのトラバースを開始する。ルートノード13の左右のブランチ両方がアクティブであるので、モジュール500は“over”演算子命令を生成する。同じ理由で、モジュール500は、バイナリノード12とバイナリノード0～5をなぞる。モジュール500が上から下へ、左から右へという方式でなぞるので、最初にバイナリノード12がなぞられる。このバイナリノード12では、バイナリ12の左ブランチが非アクティブであるので、モジュール500は“over”演算を生成しない。同じ理由で、モジュール500は左ブランチであるバイナリノード11をなぞらない。しかしながら、バイナリノード12の右ブランチがアクティブであり、バイナリノード12の左ブランチが非アクティブであり、“over”演算子に関して(/L∩R)=TRUEであるので、モジュール500はバイナリのノード8をなぞる。このバイナリノード8では、その右ブランチが非アクティブであるため、モジュール500は“out”演算を生成しない。同じ理由で、モジュール500は右ブランチであるリーフノード6をなぞらない。しかしながら、バイナリノード8の左ブランチがアクティブであり、バイナリノード8の右ブランチが非アクティブであり、“out”演算子に関する(L∩/R)=TRUEが成立するので、モジュール500はリーフノード7をなぞる。リーフノード7では、モジュール500がリーフ値命令Cを生成する。次に、リーフノード0－5をなぞるために、モジュール500はルートノード13に戻る。リーフノード0－5では、モジュール500がリーフ値命令“PAGE”を生成する。図19と図20を比較すると、モジュール500が、スキャンされた画素位置のその現在のグループに対する表現ツリーに対応する最小量の命令セットを生成していることがわかる。そして、モジュール500は、次の隣接しているエッジの間のスキャンされた画素位置の次のグループのために動作を繰り返す。

【0113】アクティビティ判断及び命令生成モジュール500の各部の動作について、図5を参照して更に詳細に説明する。

【0114】上述したように、レベルアクティベーションテーブル生成部502はレベルアクティベーションテーブルを生成する。図21をみると、そのような生成されたレベルアクティベーションテーブルの例が示されている。この特定のLevel Activation Tableは図19で示される表現ツリーのリニアライズされた書式を表す。図21のレベルアクティベーションテーブルは、表現ツリーのそれぞれのノードに関するレコードを有する。これらのレコードは、それぞれ以下のフィールドを有する。“Index”、“L Active”、“R Active”、“(L∩/R)”、“(/L∩R)”、“Leaf/Operator Entry”、“Node Active”、“Parent”、“Node is L”、“Generate L”、“Generate R”、“(L∩R) op used”、および“R Branch Index”がある。“Index”、“(L∩/R)”、“(/L∩R)”、“Leaf/Operator Entry”、“Parent”、“Node is L”、および“R Branch Index”のフィールドの内容は、現在のスキャンされた画素位置に関する関数として変化するものではないので、静的である。一方、“L Active”、“R Active”、“Node Active”、“Generate L”、“Generate R”、および“(L∩R) op used”というフィールドの内容は、現在スキャンされた画素位置によって異なる可能性がある。後者のフィールドは、隣接しているエッジの間の画素位置の各グループのために、レベルアクティベーションテーブル更新モジュール504によってアップデートされる。

【0115】“Index”フィールドは適切なレコードと関連するノードの数値ラベルを含んでいる。例えば、図19のノード12に対応するレコードは、その“Index”フィールドが12に設定される。“Parent”フィールドはレコードと関連するノードの親ノードの数値ラベルを含んでいる。例えば、図19のノード12に対応す

るレコードは「親」フィールドを１２に設定させる。すなわち、ノード１２の親ノードはノード１３である。“R branch Index”フィールドはそのレコードに関連するノードの右ブランチにぶら下がるノードの数値ラベルを含んでいる。例えば、図１９のノード１２に対応するレコードは、８に設定された“R branch Index”フィールドを有する。すなわち、ノード１２の右ブランチ上のノードはノード８である。“Node is L”フィールドは、レコードと関連する当該ノードが、その親ノードの左ブランチからぶら下がっているかどうかを示す論理フィールドである。例えば、ノード１２に対応するレコードは、TRUE(1)にセットされた“Node is L”フィールドを有する。すなわち、ノード１２は親ノード１３の左ブランチを通して親ノード１３にぶら下がっている。“Leaf/Operator Entry”フィールドは、そのレコードに関連するバイナリノードの２項演算子か、或いはリーフノードのオブジェクトを含んでいる。例えば、図１９のバイナリノード１２に対応するレコードは、合成演算子“over”にセットされた“Leaf/Operator Entry”フィールドを有する。また、別の例では、図１９のリーフノード７に対応するレコードは、オブジェクト“C”がセットされた“Leaf/Operator Entry”フィールドを有する。このようにして、表現ツリーの構造を完全に再建することができる。

【０１１６】“L Active”フィールドは、現在スキャンされている画素のグループに依存して、レコードに対応するバイナリノードの左ブランチがアクティブであるかどうかを示す論理フィールドである。同様に、“R Active”フィールドは、現在スキャンされている画素のグループに依存して、レコードに対応するバイナリノードの右ブランチがアクティブであるかどうかを示す論理フィールドである。初期化処理段階の間、スキャン中の画素は存在しないので、バイナリレコードのすべての“L Active”と“R Active”フィールドがFALSE(0)にセットされる。ただし、ルートノードの“R Active”フィールドには例外がある可能性がある。例えば、図１９のルートノード１３に対抗するレコードの“R Active”フィールドはTRUE(1)に設定される。ノード１３の右ブランチが画素のスキャンの如何にかかわらずいつもアクティブだからである。なお、リーフノードはいかなるブランチも有さないので、対応するレコードにおける“L Active”と“R Active”フィールドを設定する必要がない。

【０１１７】“(L∩/R)”フィールドは、この領域(L∩/R)からのデータが表現ツリーに渡される必要があるかどうかを示す論理フィールドである。同様に、“(/L∩R)”フィールドは、この領域(/L∩R)からのデータが表現ツリーに渡される必要があるかどうかを示す論理フィールドである。以上のように、これは合成演算子の固有の特性である。例えば、図１９のノード１２に対応するレコードは、“over”に設定された“Leaf/Operator Entry”を有する。また、(/L∩R)と(L∩/R)の領域の両方からのデータがツリーに渡される必要があるので、TRUE(1)に設定された関連するフィールド“(/L∩R)”と“(L∩/R)”を有する。別の例では、図１９のノード１１に対応するレコードは、“in”に設定された“Leaf/Operator Entry”を有する。また、領域(/L∩R)と(L∩/R)からのデータがツリーに渡される必要がないので、FALSE(0)に設定された関連するフィールド“(/L∩R)”と“(L∩/R)”を有する。これらのフィールドはバイナリノード（すなわち、合成演算）のみに関連するので、リーフノードに対応するレコードについてはこれらのフィールドを設定する必要はないことに注意されたい。

【０１１８】以下の条件、

(L∩/R && L active && !R active)

||

(L active && R active)

||

(/L∩R && !L active && R active)

==TRUE

を満足する場合に、“Node Active”フィールドは、TRUE(1)に設定され、他の場合にはFALSE(0)に設定される。

【０１１９】例えば、図１９で示されるバイナリノード８、１１、及び１２の“Node Active”フィールドは、初期化処理段階の間、FALSE(0)に設定される。すべての対応するブランチ(L ActiveとR Active)が非アクティブ（すなわち、FALSE）だからである。初期化処理段階の間、図１９のバイナリノード１３のための“Node Active”フィールドはTRUE(1)にセットされる。その右ブランチがアクティブであり（R Active=TRUE）、その左ブランチが非アクティブであり（L Active=FALSE）、その領域が演算子“over”の演算に必要であるからである。このフィールドがバイナリノード（すなわち、合成演算）だけに関連するので、リーフノードに対応するレコードのためのこれらフィールドを設定する必要がないことに注意すべきである。レコードの“Node Active”フィールドは、対応する２項演算子がアクティブであるかどうかを示す。

【０１２０】“Generate L”フィールドは、以下の条件、

(L∩/R && L active && !R active)

||

(L active && R active)

==TRUE

を満たす場合にTRUE(1)に設定され、そうでない場合にFALSE(0)に設定される論理フィールドである。

【０１２１】例えば、図１９のすべてのバイナリノード１３、１２、１１に関する“Generate L”フィールドは、初期化処理段階の間、FALSE(0)に設定される。この条件がこれらのノードのいずれによっても満たされないからである。すなわち、ノード８、１１、１２及び１３

の左ブランチがすべて非アクティブ、すなわちL Active = FALSE (0)である。また、このフィールドがバイナリノード（すなわち、合成演算）だけに関連するので、リーフノードに対応するレコードに対してこれらのフィールドを設定する必要ない。

【０１２２】同様に、“Generate R”フィールドは、以下の条件、

(L active && Ractive)

||

(/L∩R && !L active && R active)

＝＝TRUE

が満たされるときにTRUE(1)に設定され、そうでない場合にはFALSE(0)に設定される論理フィールドである。

【０１２３】例えば、初期化処理段階の間、上記条件が図１９のバイナリノード１２、１１及び８のいずれによっても満たされないので、これらのノードに対する“Generate R”フィールドはFALSE(0)に設定される。例えば、ノード８、１１及び１２の右ブランチはすべて非アクティブ、すなわちR Active=FALSE(0)である。しかしながら、バイナリノード１３の右のブランチがアクティブであるので、バイナリノード１３の“Generate R”フィールドは初期化処理の間TRUE(1)にセットされ、左のブランチは非アクティブにセットされ、(/L∩R)=TRUEである。また、このフィールドがバイナリノードだけ（すなわち、合成演算）に関連するので、リーフノードに対応するレコードにおけるこれらのフィールドを設定する必要はないことに注意されたい。

【０１２４】“(L∩R) op used”フィールドは、以下の条件、

(L active && Ractive)＝＝TRUE

が満たされるときにTRUE(1)に設定され、他の場合にはFALSE(0)に設定される論理フィールドである。

【０１２５】例えば、初期化処理段階の間、図１９のバイナリノードの全ての左ブランチが非アクティブであるので、すべてのバイナリノードに対するこのフィールドはFALSE(0)に設定される。

【０１２６】次に、図２２を参照すると、レベルアクティベーションテーブルの前述のフィールド、“Node Active”、“Generate L”、“Generate R”、および“(L∩R)op used”を設定するための論理回路が示されている。この論理回路は従来形式の一連の論理ゲートで示されており、その動作は自明であるため、これ以上の説明を省略する。

【０１２７】次に、図5に戻って、レベルアクティベーションテーブル生成モジュール５０２は初期化処理段階の間に作り出されるレベルアクティベーションテーブルをメモリ３４に格納する。次に、更新モジュール５０４は、隣接しているエッジの間のスキャンされている画素位置の各グループに対するこの初期化されたレベルアクティベーションテーブルを取得して、そのテーブルのフィールドを更新する。更新モジュール５０４は、現在のスキャンされている画素位置のグループに依存して、フィールド“L Active”、“R Active”、“Node Active”、“GenerateL”、“Generate R”、および“(L∩R) op used”の状態を変える。更新モジュール５０４は、所定の方法でレコードを更新する。その更新は、その画素位置においてアクティブなオブジェクトに対応するリーフノードの親ノードのレコードから始める。レコードにおける“Node Active”フィールドの状態を変えるときには、“Node is L”フィールドの状態に依存して、更新モジュール５０４は親ノードのレコードにおける“L active”或いは“R active”フィールドの状態における変更を引き起こす。そして、この新たに変えられた“L Active”或いは“R Active”フィールドに従って、更新モジュール５０４は親ノードのレコードの残りのフィールドを更新する。この更新プロセスは、“Node Active”フィールドが変わらずに残るところのレベルまで継続する。１つ以上のアクティブオブジェクトがあるとき、更新モジュールは一度に１つのアクティブオブジェクトについてテーブルをアップデートする。

【０１２８】更新モジュール５０４の更新プロセスについて、図２３を参照して説明する。図２３は、図２０に示したオブジェクトCがアクティブである表現ツリーのための更新されたレベルアクティベーションテーブルを示す。更新モジュール５０４は、最初に、オブジェクトCがスキャンされた画素位置の現在のグループに対してアクティブであるかを判断し、次にメモリ３４から初期化されたレベルアクティベーションテーブルを取得する。更新モジュール５０４は、リーフノードCのレコードの“Parent”フィールドから、アクティブオブジェクトCの親ノードを決定する。次に、更新モジュールは、以下の方法でテーブルのレコードを更新するべく処理を進める。これは、オブジェクトCの親ノードより開始される（レコード８）。

【０１２９】－親ノード（レコード８）の“L Active”フィールドが設定される;

－これはレコード８における“Generate L”フィールドをアサートする;

－レコード８の“Node Active”フィールドが設定される;

－親ノード（レコード１２）の“R active”フィールドが設定される;

－これはれコード１２における“Generate R”フィールドをアサートする;

－レコード１２の“Node Active”フィールドが設定される;

－親ノード（レコード１３）の“L Active”フィールドが設定される;

－これはレコード１３における“Generate L”と“L∩R op used”フィールドをアサートする;

ーレコード１３における“Node Active”フィールドが既に設定されているので、処理を止める。

【０１３０】この時点で、テーブルは命令生成のための正しい状態となる。

【０１３１】図５に戻り、更新モジュール５０４は現在スキャンされている画素位置のグループのための、この更新されたレベルアクティベーションテーブルをメモリ３４に格納する。そして、このメモリ３４より、トラバーサル及び命令生成モジュール５０６によって、更新されたレベルアクティベーションテーブルが取得される。トラバーサル及び命令生成モジュール５０６はスタックマシンベースのロボットであり、表現ツリーをトップダウン型でなぞり、更新されたレベルアクティベーションテーブルに従って命令を生成する。スタックは、レコードの“R branchIndex”を格納するのに使用される。以下の中間コードに従って、スタックマシンは以下の動作を実行する。

【０１３２】

```
IF Node is operation（ノードが演算子）
THEN
  IF L∩R op used flag is set（“L∩R op used”フラグがセット状態）
  THEN
    add operation to instruction list（演算を命令リストに追加）
  ENDIF
  IF Generate R flag is set（“Generate R”フラグがセット状態）
    Push R branch index on the stack（スタックのRブランチインデックスを Push）
  ENDIF
  IF Generate L flag is set（“Genorate L”フラグがセット状態）
    Process next entry（次のエントリへ処理を進める）
  ENDIF
ELSE (Node is a leaf)　（ノードはリーフ）
  Add leaf value instruction to instruction list（リーフ値命令を命令リストに追加）
ENDIF
IF stack is empty（スタックが空）
THEN
  STOP
ELSE
  Pop index off stack（スタックからインデックスをポップ）
  Process the entry （エントリを処理）
ENDIF
```

【０１３３】命令モジュール５０６のトラバースと命令生成プロセスについて、図２３を参照して説明する。図２３は、オブジェクトCがアクティブである図２０の表現ツリーのための更新されたレベルアクティベーションテーブルを示す。上記の中間コードと図２３のレベルアクティベーションテーブルとに従った命令の生成は以下のように進行する。

【０１３４】（i）処理はレコード１３（ルートノード）より開始する；

（ii）レコード１３の“L∩R op used”フィールドがTRUE(1) なので、“over”命令が命令リストに加えられる；

（iii） レコード１３の“Generate R”フィールドがTRUE(1)なので、レコード１３の“R Branch Index”の内容がスタックに加えられる。その結果、スタックはインデックス（５）を格納する；

（iv）レコード１３の“Generate L”フィールドがTRUE(1)であるので、処理はレコード１２に進む；

（v）レコード１２の“L∩R op used”フィールドがFALSE(0)であるので、いかなる命令も加えられない；

（vi） レコード１２の“Generate R”フィールドがTRUE(1)であるので、レコード１２の“R Branch Index”の内容がスタックに加えられる。その結果、スタックは、その時点で、インデックス（８，５）を格納する；

（vii） レコード１２の“Generate L”フィールドがFALSE(0)であり、その結果、処理は抜け落ちる；

(viii) スタックが空ではないので、インデックス８がスタックからポップされて、処理はインデックス８に対応するレコードへ移動する。その結果、この時点で、スタックはインデックス（５）を格納することになる；

（ix）レコード８の“L∩R op used”フィールドがFALSE(0)であるので、どんな命令も加えられない；
（x）レコード８の“Generate R”フィールドがFALSE(0)であるので、スタックに何も加えられない。その結果、スタックは、この時点でインデックス（５）を格納する；
（xi）レコード８のフィールド“Generate L”がTRUE(1)であるので、処理は次のエントリ、すなわちレコード７に移動する；
（xii）レコード７がリーフであるので、リーフ値命令がリストに加えられて、処理は抜け落ちる；
（xiii）スタックが空でないので、インデックス５がスタックからポップされて、処理はインデックス５へ移動する。スタックはこの時点で空になる。このプロセスは完了へと続く。
【０１３５】この例において生成された命令が図２０に示される。
【０１３６】このようにして、トラバーサル及び命令生成モジュール５０６は、あらゆる１つ以上のアクティブオブジェクトのために、表現ツリーの描画に必要な命令の最小限のセットを生成することができる。これらの命令は、次に、色塗潰し決定モジュール６００に渡される。
【０１３７】アクティビティ判断及び命令生成モジュール５００の別の実施形態において、モジュールは、不透明物体がオーバラッピング領域でもう片方のオブジェクトを見えなくするかもしれないという事実を考慮に入れる。実際に、ノードのブランチの一方は、他方によって隠され得る。両方のオブジェクトがアクティブである場合に、オブジェクトが不透明物体によって隠されるならば、そのオブジェクトを生成する必要はない。これらの余分な処理が実行されるのを防ぐために、レベルアクティベーションテーブルを変更することができる。
【０１３８】次に図２５を参照すると、そのような改造されたレベルアクティベーションテーブルの例が示されている。このテーブルは、２つのフィールド“L obscures R（LはRを隠す）”と“R obscures L（RはLを隠す）”が加わっているという点を除いて、図２１のレベルアクティベーションテーブルと同じ構造である。“LはRを隠す”フィールドは論理フィールドであり、左右両方のオブジェクトがアクティブであるときに、左のオブジェクトが右のオブジェクトを隠すかどうかを示す。同様に、フィールド“RはLを隠す”も論理フィールドであり、左右両方のオブジェクトがアクティブであるときに、右のオブジェクトが左のオブジェクトを隠すかどうかを示す。“Generate R”、“L∩R op used”及び“Generate L”フィールドに含まれるデータも、図２１のレベルアクティベーションテーブルとは異なる論理に基づく。obscuranceフラグは不透明なオブジェクトと透明なオブジェクトを含む表現ツリーで非常に役に立つ。
【０１３９】図２４には、改造されたレベルアクティベーションテーブルのフィールド、“Node Active”、“Generate L”、“Generate R”、および“(L∩R) op used”を設定するための論理回路が示される。図２２に対して“Node Active”の論理が変化しており、その時点のデータがノードによって生成されたかどうかに基づいていることがわかる。回路のクリティカルパスは長いが、これは、同じ論理を使用して実行されるオブジェクトのクリッピングを可能とする。obscuranceフィールドの両方がセットされるならば、どんなデータも論理積領域において生成されない（これは、CLIP OUTの実現の一部である）。
【０１４０】これらのobscuranceフィールドは、図２５のLevel Activation Tableに示されるように、CLIP IN動作とCLIP OUT動作を実現するのに用いられてもよい。CLIPINとCLIP OUT動作のためのフィールド設定は図２５のTableに示されている。
【０１４１】CLIP IN動作と単純なin動作との間の違いは、右ブランチ動作が決して実行されないということである。図２６に示されるIN動作の真理値表と図２７に示されるCLIP IN動作の真理値表との比較からこれを見ることができる。これらの真理値表では、Tは透明(非アクティブ)を表し、Oは不透明を表す。そして、LとRはそれぞれLの不透明が未知であること、Rの不透明が未知であることを表す。違いのあるボックスは強調表示されている。CLIP INの場合では、右オブジェクトの使用開始が左オブジェクトのための命令の生成を可能にするので、左のオブジェクトは切り取られるエッジである。なお、右オブジェクトは任意の多くのオブジェクトの合成であってもよい。この合成には、必要に応じたいかなる組合せ、異なる充てんルールも使用してよい。右オブジェクトのアクティビティ状態は、全てCLIP IN動作によって使用される。従前のシステムでは、切取りオブジェクトは別々の単一レベルから成る。それは、アクティベートされるかまたはディアクティベートされるときに、クリップが適用されるすべてのレベルに対するカウンタに作用した。本構成においては、クリップオブジェクトは特殊なものとして全体図オブジェクトの特殊ケースとして現れ、表現ツリーのノードとして現れる。そして、その結果、あらゆるレベルでカウンタを増加するかまたは減少させる必要性なく、アクティブ化或いは非アクティブ化がクリップを適用する。
【０１４２】CLIP OUT動作と単純なout動作との間の違いは、図２８に示されるＯＵＴ動作の真理値表と図２９に示されるCLIP IN動作に関する真理値表との比較から見ることができる。ここでも、Tは透明（非アクティブ）を表し、Oは不透明を表し、LとRは、それぞれLの不透明が未知であること、Rの不透明が未知であることを表す。また、違いの生じているボックスが強調表示されている。CLIP OUTの場合では、右オブジェクトのア

クティブ化が左オブジェクトのための命令の生成を防ぐので、左オブジェクトは切り取られるエッジである。CLIP INに関しては、右オブジェクトは必要に応じた異なった充てんルールを有する、オブジェクトの任意の複雑なコレクションとなるかもしれない。クリッピングに使用されるのは、アクティビティの状態だけである。

【０１４３】上述したアクティビティ判断及び命令生成モジュール５００は、表現ツリーに基づく。しかしながら、モジュール５００の本質はＤＡＧｓ（有向隣接図表：Directed Adjacency Graphs）に一般化され得る。これは、レベルアクティベーションテーブルのレコードの親ノードフィールドが、その親ノードへのテーブルエントリのリストを含むことを許容し、Ｌインデックスポインタを提供することによって、モジュール５００の更なる実施形態によって達成され得る。ノードの状態の変更は、その親ノードのすべてが変更されることを要求し、命令の生成は、単に次のテーブルエントリを探すのではなく、Ｌインデックスに従うために必要となる。ＤＡＧｓは、クリッピングオブジェクトとの使用に関して有用である。そこでは、複数のオブジェクトが同じオブジェクトによってクリップされる。

【０１４４】更に別の実施形態は、開始ノード、すなわち、演算或いはデータを提供する最初のノード、がその左ブランチからのデータをレンダリングされたページ上に合成するということに基づく。これは、左ブランチをレンダリングされたページと合成するノードを特定するレベルアクティベーションテーブルに“Page”フィールドを加えることによって特定され得る（R is PAGE）。そのページにデータを置いているアクティブなノードを特定するためのテーブルエントリデータを１ビットで提供するために、この“Page”フィールドと“Generate L”フィールドとをANDしてもよい。開始ノードはこれらのうちの最優先ノードであり、命令生成のために、サーチされ得る。これはツリーの幹にある、動作を提供しないノードに関してツリーをなぞることによるオーバヘッドを排除する。

【０１４５】次に、図３０をみると、レンダリングされたページにそれらの左ブランチを合成するノードを示す模式的な表現ツリー３００が示されている。わかるように、ノード３０２、３０４、３０６、および３０８はそれらの左ブランチがレンダリングされたブランチ上に合成され、これらのノードの各々の“Page”フィールドがTRUE(1)にセットされる。この“Page”フィールドは同じノードの“GenerateL”フィールドとのANDであってもよい。ここで、Generate L”フィールドは、ページにデータを置くそれらのアクティブなノードを確認する。表現ツリーの例において、ノード３０２、３０６、および３０８はアクティブであって、ノード３０４は非アクティブである。したがって、必要な命令を生成するために、アクティブなノード３０２、３０６、および３０８のそれらのサブツリーだけがなぞられる必要がある。こうして、ツリーのトラバーサルを最小にする、以上、好適な実施形態としてアクティビティ判断及び命令生成モジュール５００を集積回路として説明したが、ホストプロセッサ２等のような汎用処理ユニット上で実行可能な等価なソフトウエアモジュールとして実現されてもよい。ソフトウェアモジュールはディスク装置やコンピュータネットワークなどのようなコンピュータ可読媒体を介してユーザに配られ得るコンピュータプログラム製品の一部を形成してもよい。

【０１４６】４．０　塗潰し色決定モジュール

塗潰し色決定モジュール６００の動作を、図６を参照して、以下に説明する。アクティビティ判断及び命令生成モジュール５００からの着信メッセージ５９８は、塗潰しデータ設定メッセージ、反復メッセージ、塗潰し優先順位メッセージ、画素の終りメッセージおよびスキャンラインの終りメッセージを含み、まず、塗潰しルックアップ及び制御モジュール６０４に渡される。塗潰しルックアップ及び制御モジュール６０４は、塗潰し色決定モジュール６００のさまざまな構成要素による使用のために、現行Ｘ位置カウンタ６１４および現行Ｙ位置カウンタ６１６を維持する。

【０１４７】スキャンラインの終りメッセージを受け取った時には、塗潰しルックアップ及び制御モジュール６０４は、現行Ｘ位置カウンタ６１４を０にリセットし、現行Ｙ位置カウンタ６１６をインクリメントする。スキャンラインの終りメッセージは、その後、画素合成モジュール７００に渡される。

【０１４８】塗潰しデータ設定メッセージを受け取ったときには、塗潰しルックアップ及び制御モジュール６０４は、塗潰しデータテーブル３６の指定された位置６０２にそのデータを記憶する。

【０１４９】反復メッセージを受け取ったときには、塗潰しルックアップおよび制御モジュール６０４は、反復メッセージからのカウントだけ現行Ｘ位置カウンタ６１４をインクリメントする。反復メッセージは、その後、画素合成モジュール７００に渡される。

【０１５０】画素の終りメッセージを受け取ったときには、塗潰しルックアップ及び制御モジュール６０４は、やはり現行Ｘ位置カウンタ６１４をインクリメントし、この画素の終りメッセージは、その後、画素合成モジュール７００に渡される。

【０１５１】塗潰し優先順位メッセージを受け取ったときには、塗潰しルックアップ及び制御モジュール６０４は、下記の処理を含む動作を実行する。

【０１５２】（ｉ）塗潰し優先順位メッセージからの塗潰しタイプを使用して、テーブル３６のレコードサイズを選択する、（ii）塗潰し優先順位メッセージからの塗潰しテーブルアドレスと、上で決定されたレコードサイズを使用して、塗潰しデータテーブル３６からレコード

を選択する、（iii）塗潰し優先順位メッセージからの塗潰しタイプを使用して、塗潰し色の生成を実行するサブモジュールを決定し、選択する。サブモジュールには、ラスタイメージモジュール６０６、フラットカラーモジュール６０８、リニアランプカラーモジュール６１０および不透明タイルモジュール６１２を含めることができる、（iv）決定されたレコードを、選択されたサブモジュール６０６乃至６１２に供給する、（ｖ）選択されたサブモジュール６０６乃至６１２が、供給されたデータを使用して、色と不透明度の値を決定する。

【０１５３】（vi）決定された色と不透明度は、塗潰し色メッセージからの残りの情報、すなわち、ラスタ演算コード、アルファチャネル演算コード、ソースポップフラグおよびデスティネーションポップフラグと組み合わされて、色合成メッセージを形成し、この色合成メッセージは、接続６９８を介して画素合成モジュール７００に送られる。

【０１５４】好ましい実施形態では、決定された色と不透明度が、８ビットの精度の赤、緑、青および不透明度の４つの組であり、通常の形で３２ビット／画素が与えられる。しかし、シアン、マゼンタ、黄および黒の、不透明度を含有する４つの組か、他の多数の既知の色表現のうちの１つを、その代わりに使用することができる。以下では、赤、緑、青および不透明度の場合を説明するが、この説明は、他の場合にも適用することができる。

【０１５５】ラスタ画像モジュール６０６、フラットカラーモジュール６０８、リニアランプカラーモジュール６１０および不透明タイルモジュール６１２の動作を以下に説明する。

【０１５６】フラットカラーモジュール６０８は、供給されたレコードを、３つの８ビットの色成分（通常は赤、緑および青の成分と解釈される）と１つの８ビットの不透明度値（通常は、指定された色によって覆われる画素の割合の尺度と解釈され、０は覆われないすなわち完全な透明を意味し、２５５は完全に覆われるすなわち完全な不透明を意味する）を含む固定フォーマットのレコードと解釈する。この色と不透明度の値は、接続６９８を介して直接に出力され、それ以上の処理なしで、決定された色および不透明度を形成する。

【０１５７】リニアランプカラーモジュール６１０は、供給されたレコードを、３つの色成分および１つの不透明度成分に関連する４組の定数cx、cyおよびdと、リニアランプの基準点の座標である２つの位置値xbaseおよびybaseを含む固定フォーマットのレコードと解釈する。基準点では、色成分と不透明度成分が、それに関連付けられたd値を有する。

【０１５８】４組のそれぞれについて、結果の値rは、次式を使用して、現在のXY座標とxbaseおよびybase定数に３つの定数を組み合わせることによって計算される。すなわち、

$$r=\mathrm{clamp}(cx\times(X-xbase)+cy\times(Y-ybase)+d)$$

ここで、関数clampは、

$$\mathrm{clamp}(x)=\begin{cases}255 & 255\leqq x\\ \lfloor x \rfloor & 0\leqq x<255\\ 0 & x<0\end{cases}$$

のように定義される。

【０１５９】代替の実装では、リニアランプカラーモジュール６１０は、供給されたレコードを、３つの色成分および１つの不透明度成分に関連する、４組の３つの定数cx、cyおよびdを含む固定フォーマットのレコードと解釈する。これらの４組のそれぞれについて、結果の値ｒは、次式を使用して、３つの定数を現行Ｘカウントであるｘおよび現行Ｙカウントであるｙと組み合わせることによって、

$$r=\mathrm{clamp}(cx\times x+cy\times y+d)$$

により計算される。ここで、関数clampは、上で定義されたものである。

【０１６０】以上のように生成された４つの結果は、色と不透明度の値に形成される。この色と不透明度の値は、接続６９８を介して直接に出力され、それ以上の処理なしで決定された色および不透明度を形成する。

【０１６１】同一の結果を与える他の算術計算を使用することができる。

【０１６２】不透明タイルモジュール６１２は、供給されたレコードを、３つの８ビット色成分、１つの８ビット不透明度値、整数のＸ位相（px）、Ｙ位相（py）、Ｘスケール（sx）、Ｙスケール（sy）および６４ビットのマスクを含む固定フォーマットのレコードと解釈する。これらの値は、表示リスト生成から発し、通常は、元のページ記述に含まれる。ビットマスク内のビットアドレスaは、以下の式、

$$a=((x/2^{sx}+px)\bmod 8)+((y/2^{sy}+py)\bmod 8)\times 8$$

で決定される。

【０１６３】ビットマスク内のアドレス「a」のビットが検査される。検査されるビットが１の場合には、レコードからの色と不透明度が、モジュール６１２の出力に直接にコピーされ、決定された色および不透明度を形成する。検査されるビットが０の場合には、３つの０成分値を有する色と０の不透明度値が形成され、決定された色および不透明度として出力される。

【０１６４】ラスタ画像モジュール６０６は、供給されたレコードを、６つの定数a、b、c、d、xbaseおよびybaseと、サンプリングされるラスタ画像画素データ１６の各ラスタライン内のビット数の整数カウント（bpl）と、画素タイプとを含む固定フォーマットのレコードと解釈する。画素タイプは、ラスタ画像画素データ内の画素データ１６を、次のうちのどれとして解釈するかを示す。

【０１６５】（ｉ）１ビット毎画素の白黒不透明画素
（ii）１ビット毎画素の不透明黒または透明の画素

（iii）８ビット毎画素のグレイスケール不透明画素
（iv）８ビット毎画素の黒不透明スケール画素
（ｖ）２４ビット毎画素の不透明３色成分画素
（vi）３２ビット毎画素の３色成分と不透明度の画素
他の多くのフォーマットが可能である。

【０１６６】ラスタ画像モジュール６０６は、画素タイプインジケータを使用して、ビット単位の画素サイズ（ｂｐｐ）を決定する。その後、ラスタ画像画素データ１６内のビットアドレスａを、以下の式、

a=bpp* └a×(x-xbase)+c×(y-ybase)┘ +bpl× └b×(x-xbase)+d×(y-ybase)┘

から計算する。

【０１６７】レコード６０２からの画素タイプに従って解釈された画素は、ラスタ画像画素データ１６内の計算されたアドレス「ａ」から取り出される。この画素は、３つの８ビット色成分と１つの８ビット不透明度成分を有するために、必要に応じて展開される。「展開」とは、たとえば、８ビット／画素のグレイスケール不透明ラスタ画像からの画素が、赤、緑および青の成分のそれぞれに適用されるサンプリングされた８ビット値と、完全な不透明がセットされた不透明度成分を持つことを意味する。これが、画素合成モジュール７００への決定された色および不透明度の出力６９８を形成する。

【０１６８】その結果、表示可能オブジェクト内で有効なラスタ画素データは、メモリ内の画像画素データ１６へのマッピングの決定を介して得られる。これによって、ラスタ画素データのオブジェクトベース画像へのアフィン変換が効率的に実施され、これは、グラフィックオブジェクトとの合成が発生する可能性がある場合に画像ソースからの画素データをフレームストアに転送する従来技術の方法より効率的である。

【０１６９】上記に対する好ましい特徴として、任意選択として、ラスタ画像画素データ１６内の画素間の補間を、まず中間結果ｐおよびｑを以下の式、

p=a×(x-xbase)+c×(y-ybase)
q=b×(x-xbase)+d×(y-ybase)

に従って計算することによって実行することができる。

【０１７０】次に、ラスタ画像画素データ１６内の４画素のビットアドレスa_{00}、a_{01}、a_{10}およびa_{11}を、以下の式、

a_{00}=bpp× └p┘ +bpl× └q┘
a_{01}=a_{00}+bpp
a_{10}=a_{00}+bpl
a_{11}=a_{00}+bpl+bpp

に従って決定する。

【０１７１】次に、各色成分および不透明度成分のそれぞれについて、結果の画素成分値rを、以下の式

r=interp(interp(get(a_{00}),get(a_{01}),p),interp(get(a_{10}),get(a_{11}),p),q)

に従って、決定する。ここで、関数interpは、

interp(a,b,c)=a+(b-a)×(c- └c┘)

のように定義される。

【０１７２】上の諸式で、表現 └値┘ =floor（値）であり、floor演算では、値の端数部分の切捨が行われる。

【０１７３】get関数は、所与のビットアドレスでの、ラスタ画像画素データ１６からサンプリングされた現行画素成分の値を返す。いくつかの画像タイプのいくつかの成分について、これが暗黙の値になる可能性があることに留意されたい。

【０１７４】上記に対する好ましい特徴として、任意選択として、供給されたレコードから読み取られるタイルサイズを用いる剰余演算によって現行Ｘ位置カウンタ６１４および現行Ｙ位置カウンタ６１６から導出されるｘおよびｙの値を上式で使用することによって、画像タイリングを実行することができる。

【０１７５】多数の他のこのような塗潰し色生成サブモジュールが可能である。

【０１７６】５．０　画素合成モジュール

以下に、画素合成モジュール７００の動作を説明する。塗潰し色決定モジュール６００からの着信メッセージには、反復メッセージ、色合成メッセージ、画素の終りメッセージおよびスキャンラインの終りメッセージが含まれるが、このメッセージは、順番に処理される。

【０１７７】反復メッセージまたはスキャンラインの終りメッセージを受け取った時には、画素合成モジュール７００は、それ以上の処理なしで画素出力ＦＩＦＯ７０２にそのメッセージを転送する。

【０１７８】色合成メッセージを受け取った場合の画素合成モジュール７００の動作概要を示せば、色合成メッセージからの色と不透明度を、色合成メッセージからのラスタ演算およびアルファチャネル演算に従って、画素合成スタック３８からポップされた色および不透明度と組み合わせる。その後、結果を画素合成スタック３８にプッシュする。色合成メッセージの受取り時に実行される処理の説明は以下でなされる。

【０１７９】画素の終りメッセージを受け取った時には、画素合成モジュール７００は、画素合成スタック３８から色と不透明度をポップする。ただし、スタック３８が空の場合には不透明の白の値が使用される。結果の色と不透明度は、画素出力ＦＩＦＯに転送される画素出力メッセージに形成される。

【０１８０】既知の合成アプローチは、ポーター（Porter,T）およびダフ（Duff,T）著「Compositing Digital Images」、Computer Graphics誌、Vol.18 No.3 1984年、第253～259ページに記載されている。ポーターーダフ合成演算の例を、図１５に示す。しかし、このようなアプローチでは、合成によって形成される交差領域内のソースおよびデスティネーションの色の処理だけが可能であり、その結果、交差領域内の透明度の影響には適合できないという点で不完全である。その結果、ポーター

およびダフによって定義されたラスタ演算は、透明度の存在下で基本的に動作不能である。

【０１８１】色合成メッセージを受け取った時に画素合成モジュール７００によって実行される処理を以下に説明する。

【０１８２】色合成メッセージを受け取った際に、画素合成モジュール７００は、まず、ソースの色と不透明度を形成する。これは、色合成メッセージでポップソースフラグがセットされていない限り、色合成メッセージで供給された色と不透明度が使用され、ポップソースフラグがセットされている場合には、その代わりに画素合成スタック３８から色がポップされる。この場合、すなわち、画素合成スタックのポップ中に画素合成スタック３８が空であることがわかった場合、エラー表示なしで不透明の白の色値が使用される。次に、デスティネーションの色と不透明度が、画素合成スタック３８からポップされる。ただし、色合成メッセージでデスティネーションポップフラグがセットされていない場合には、スタックポインタが「ポップ」動作中に変更されず、その結果スタック３８の最上部からの読み取りになる。

【０１８３】ソースの色および不透明度をデスティネーションの色および不透明度と組み合わせる方法を、図７Ａないし図７Ｃを参照して以下に説明する。色および不透明度の値は、通常は0から２５５までの範囲の８ビット値として記憶されるが、ここでは説明の目的のために、０から１までの範囲（すなわち正規化されている）とみなす。２画素を合成する目的のために、各画素が２つの領域、すなわち、完全に不透明な領域と完全に透明な領域に分割されるものとみなし、不透明度値は、これら２つの領域の比率の表示であるとみなされる。図７Ａに、図示されない３成分色値と、不透明度値（so）を有するソース画素７０２を示す。ソース画素７０２の影付きの領域は、画素７０２の完全に不透明な部分７０４を表す。同様に、図７Ａの影付きでない領域は、ソース画素７０２のうちで、完全に透明であるとみなされる部分７０６を表す。図７Ｂに、ある不透明度値（do）を有するデスティネーション画素７１０を示す。デスティネーション画素７１０の影付きの領域は、画素７１０の完全に不透明な部分７１２を表す。同様に、画素７１０は、完全に透明な部分７１４を有する。ソース画素７０２およびデスティネーション画素７１０の不透明領域は、組合せのために、互いに直交するとみなされる。これら２つの画素のオーバーレイ７１６を、図７Ｃに示す。対象の３つの領域が存在し、これには、so×（1-do）の面積を有するデスティネーション外ソース７１８、面積so×doを有するソースデスティネーション交差７２０および、（1-so）×doの面積を有するソース外デスティネーション７２２が含まれる。これら３つの領域のそれぞれの色値は、概念上は独立に計算される。デスティネーション外ソース領域７１８は、その色をソース色から直接にとる。ソース外デスティネーション領域７２２は、その色をデスティネーション色から直接にとる。ソースデスティネーション交差領域７２０では、その色をソース色とデスティネーション色の組合せからとる。上で説明した他の動作とは別個の、ソース色とデスティネーション色の組合せの処理は、ラスタ演算と称され、これは、画素合成メッセージからのラスタ演算コードによって指定される、１組の関数のうちの１つである。好ましい実施形態に含まれるラスタ演算の一部を、図１７に示す。

【０１８４】各関数は、ソース色およびデスティネーション色の対応する色成分の対のそれぞれに適用されて、結果の色において同様の成分が得られる。多くの他の関数が可能である。

【０１８５】画素合成メッセージからのアルファチャネル演算も考慮されている。アルファチャネル演算は、３つのフラグを使用して実行され、フラグのそれぞれは、ソース画素７０２とデスティネーション画素７１０のオーバーレイ７１６内の対象の領域のうちの１つに対応する。領域のそれぞれについて、領域の不透明度値が形成され、この不透明度値は、アルファチャネル演算で対応するフラグがセットされていない場合には０、それ以外の場合にはその領域の面積である。

【０１８６】結果の不透明度は、領域の不透明度の合計から形成される。結果の色の各成分は、領域の色と領域の不透明度の対のそれぞれの積の和を結果の不透明度で除算することによって形成される。

【０１８７】結果の色および不透明度は、画素合成スタック３８にプッシュされる。

【０１８８】６．０　画素出力モジュール

画素出力モジュール８００の動作を以下に説明する。着信メッセージは、画素出力ＦＩＦＯから読み取られ、これには、画素出力メッセージ、反復メッセージおよびスキャンラインの終りメッセージが含まれ、順番に処理される。

【０１８９】画素出力メッセージを受け取った時に、画素出力モジュール８００は、画素を記憶し、その画素をその出力に転送する。反復メッセージを受け取った時には、最後に記憶した画素が、反復メッセージからのカウントによって指定される回数だけ、出力８９８に転送される。スキャンラインの終りメッセージを受け取った時には、画素出力モジュール８００は、そのメッセージをその出力に渡す。

【０１９０】出力８９８は、必要に応じて、画素画像データを使用する装置に接続することができる。このような装置には、ビデオ表示ユニットまたはプリンタなどの出力装置、または、ハードディスク、ラインストア、バンドストアまたはフレームストアを含む半導体ＲＡＭなどのメモリ記憶装置、または、コンピュータネットワークが含まれる。

【０１９１】多数のレベルにわたるオブジェクトの合成

も、本明細書に記載の方法の外挿によって可能であることを、当業者は了解するであろう。さらに、示されたさまざまな操作を、フレームストアベースシステムおよび他のスタックベース、ラインベースまたはバンドベースの合成システムを含む異なる合成パラダイムで使用することができることも諒解されるであろう。

【０１９２】前述から、本明細書に記載の方法および装置が、レンダリング処理中の画素画像データの中間記憶を必要とせずに、洗練されたグラフィック記述言語が必要とする機能性のすべてを備えたグラフィックオブジェクトのレンダリングを提供することは、明白であろう。

【０１９３】産業上の利用性

本発明の実施形態がコンピュータグラフィックス及び印刷産業に適用可能であることが上述の説明より明らかである。

【０１９４】前述は、本発明の一つの或いはいくつかの実施形態を記述したのみであり、それらは説明のために用いられるもので限定的なものではなく、それらに対して本発明の趣旨および範囲を逸脱せずに改造及び変更を行なうことができる。

【図面の簡単な説明】

【図１】好ましい実施形態を組み込まれるコンピュータシステムの概略ブロック図である。

【図２】好ましい実施形態の機能データフローを示すブロック図である。

【図３】好ましい実施形態の画素シーケンシャルレンダリング装置と、関連する表示リストおよび一時ストアの概略ブロック図である。

【図４】図２のエッジ処理モジュールの概略機能図である。

【図５】図２の優先順位決定モジュールの概略機能図である。

【図６】図２の塗潰しデータ決定モジュールの概略機能図である。

【図７】ソースとデスティネーションの間の画素の組合せを示す図である。

【図８】好ましい実施形態の演算を説明するための例として使用される、オブジェクトが２つある画像を示す図である。

【図９】図８のオブジェクトのベクトルエッジを示す図である。

【図１０】図８の画像の複数のスキャンラインのレンダリングを示す図である。

【図１１】図８の画像のエッジレコードの配置を示す図である。

【図１２Ａ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｂ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｃ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｄ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｅ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｆ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｇ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｈ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｉ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１２Ｊ】図１０の例のために図４の配置によって実施されるエッジ更新ルーチンを示す図である。

【図１３Ａ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１３Ｂ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１３Ｃ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１３Ｄ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１３Ｅ】Ｘ座標の大きな変化がスピル条件にどのように寄与し、それらがどのように処理されるかを示す図である。

【図１４】２つの従来技術のエッジ記述フォーマットと、好ましい実施形態で使用されるエッジ記述フォーマットを比較するための図である。

【図１５】いくつかの合成演算の結果を示す図である。

【図１６】図２のエッジレコード格納部における格納のための、エッジレコードの模式的なテーブルを示す図である。

【図１７】好適な実施形態において用いられるラスタ演算子のいくつかに関するテーブルを示す図である。

【図１８】エクスプレッションツリーとして示される単純な合成式と対応する図を示す図である。

【図１９】オブジェクトＡ，Ｂ，Ｃ及びＤ、及び対応する命令リストに関して、一連のPorterとDuffの合成操作を実現するための典型的な表現ツリーを示す図である。

【図２０】オブジェクトＣが有効な場合の図１９の表現ツリーと対応する命令リストを示す図である。

【図２１】図５に示したレベルアクティベーションテーブル生成部によって生成される、図２０の表現ツリーに関するレベルアクティベーションテーブルを示す図である。

【図２２】図５に示したレベルアクティベーションテーブルの更新モジュールにおいて使用される論理回路を示す図である。

【図２３】オブジェクトＣが有効である、図２０の表現ツリーのための更新されたレベルアクティベーションテーブルを示す図である。

【図２４】図５に示したレベルアクティベーションテーブルの更新モジュールにおいて使用される論理回路の別の具体例を示す図である。

【図２５】図２４の更なる実施形態と共に用いるためのレベルアクティベーションテーブルを示す図である。

【図２６】ＩＮ合成操作のための真理値表を示す図である。

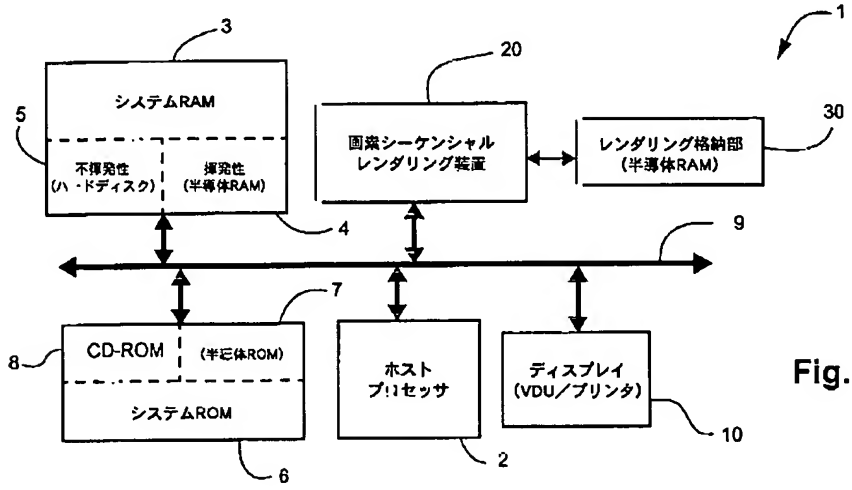
【図２７】ＣＬＩＰ　ＩＮ合成操作のための真理値表を示す図である。

【図２８】ＯＵＴ合成操作のための真理値表を示す図である。

【図２９】ＣＬＩＰ　ＯＵＴ合成操作のための真理値表を示す図である。

【図３０】左側のブランチを描画されたページに合成するノードを支援す表現ツリーの典型的な例を示す図である。

【図１】



【図２】

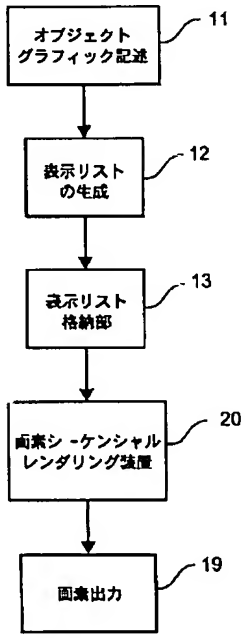


Fig. 1

Fig. 2

【図８】

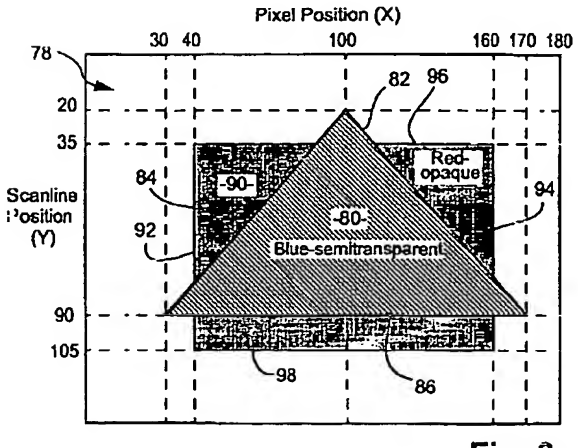


Fig. 8

【図１０】

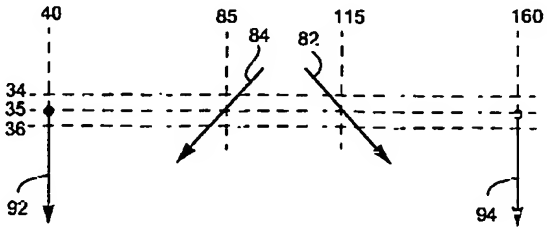


Fig. 10

【図3】

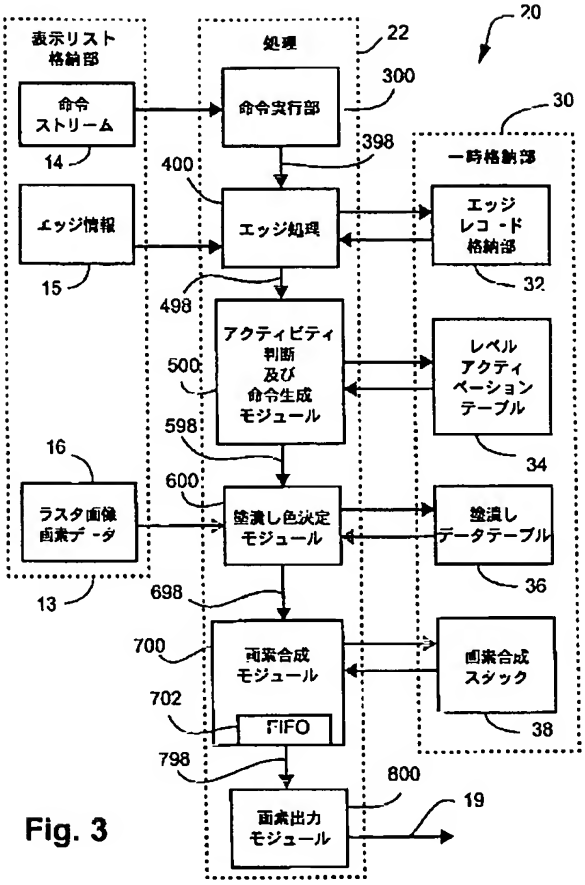
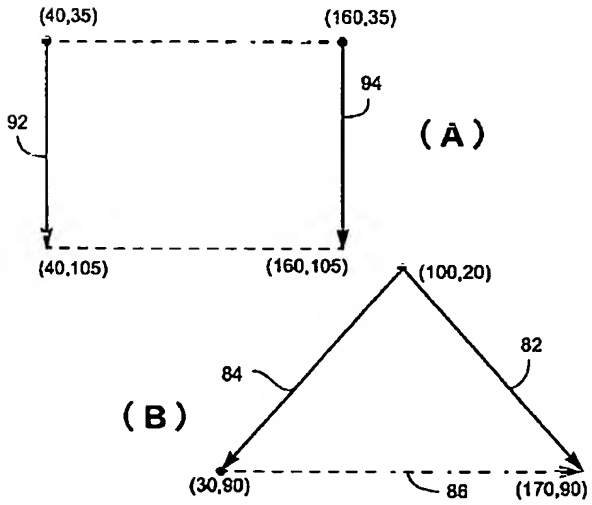


Fig. 3

【図9】

Fig. 9



【図5】

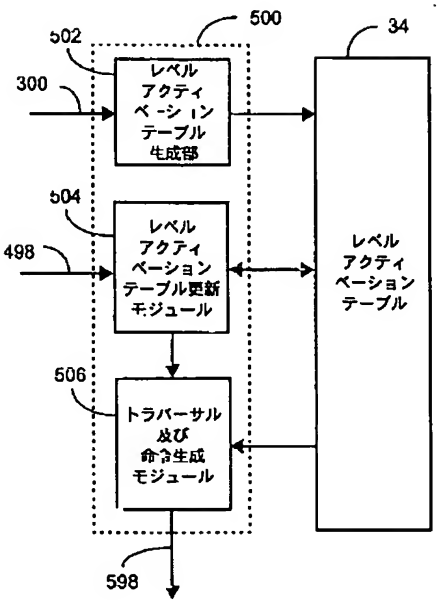


Fig. 5

【図11】

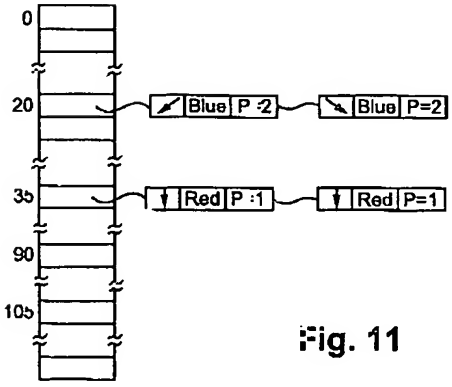


Fig. 11

【図12A】

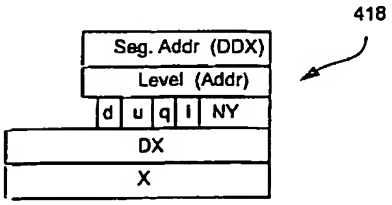


Fig. 12A

【図4】

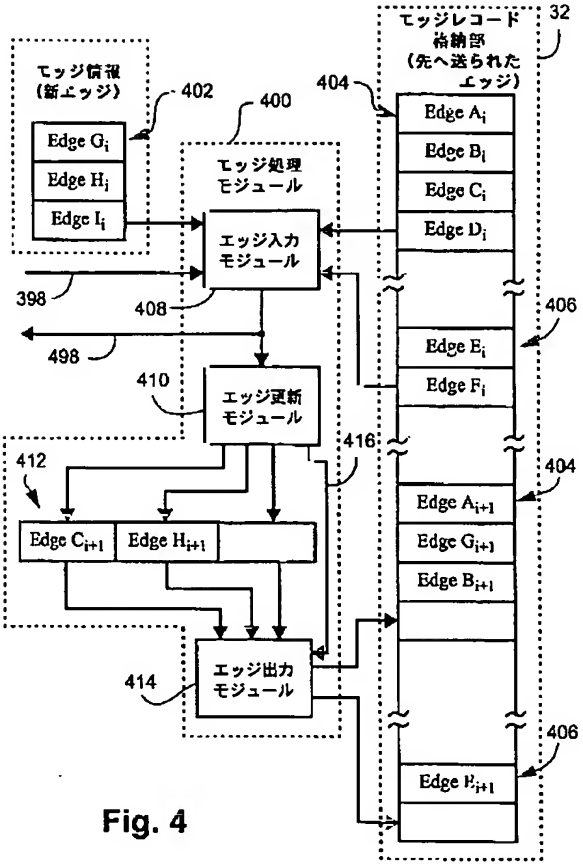
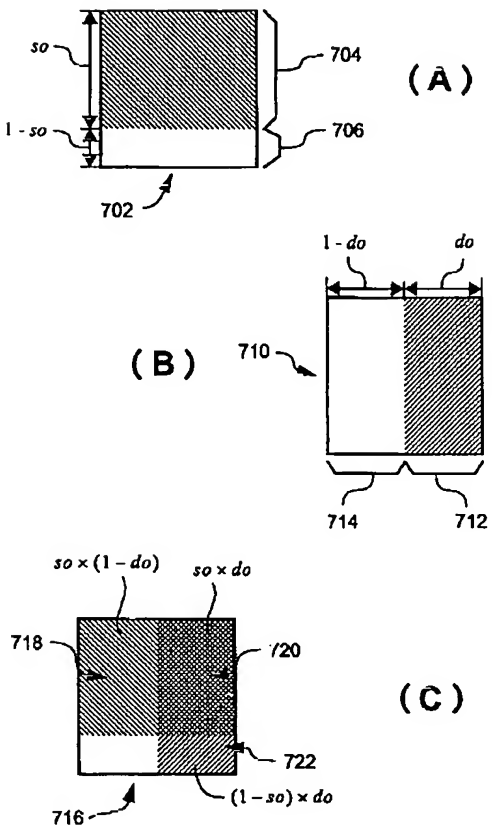


Fig. 4

【図7】

Fig. 7



【図12B】

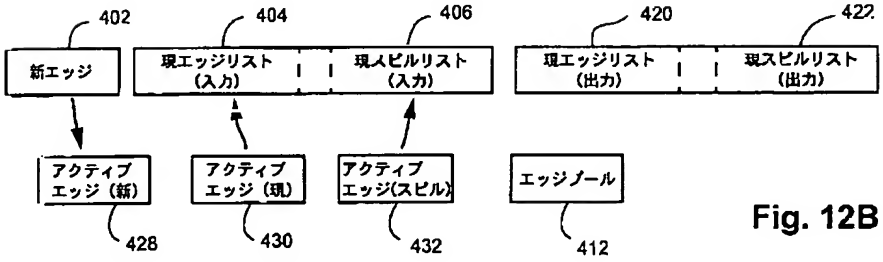


Fig. 12B

【図12C】

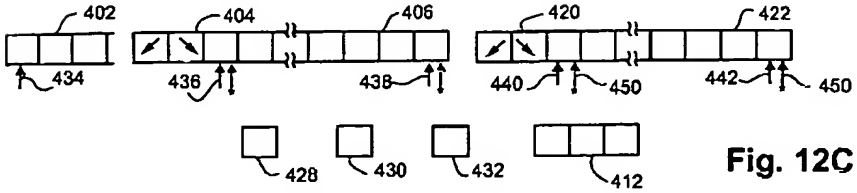


Fig. 12C

【図6】

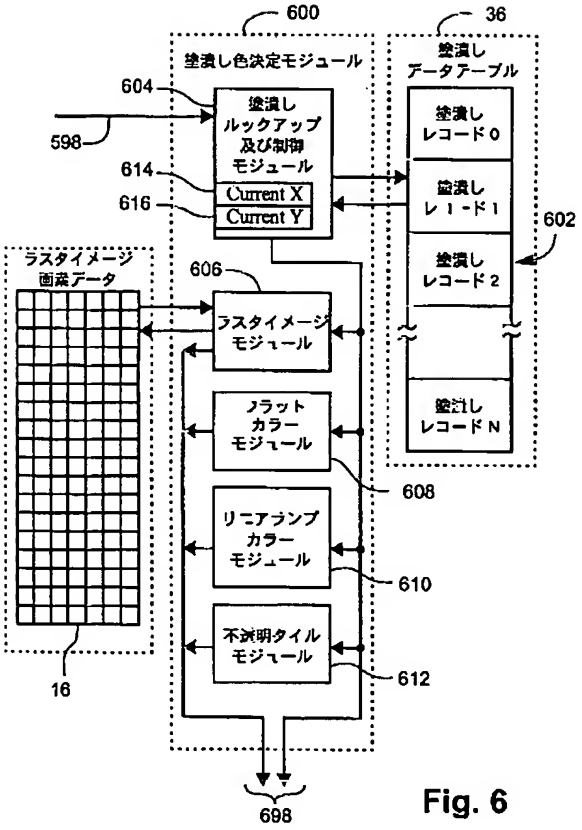


Fig. 6

【図13A】

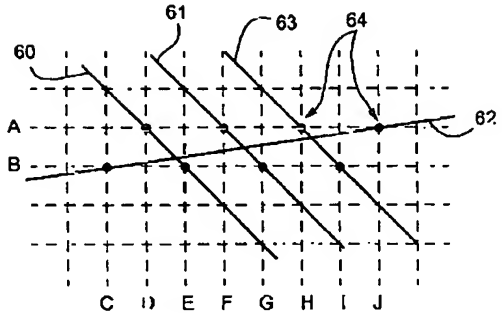


Fig. 13A

【図12D】

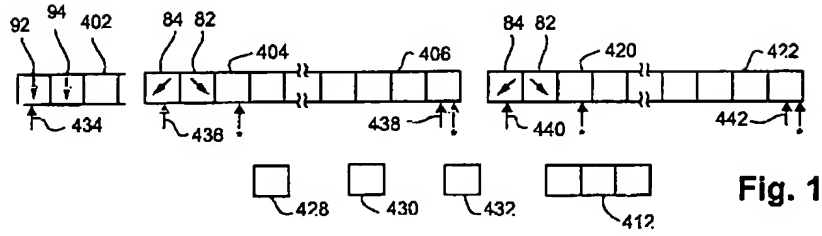


Fig. 12D

【図12E】

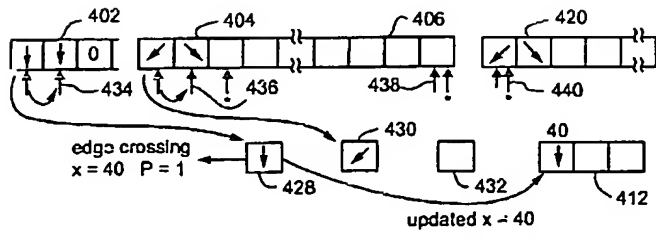


Fig. 12E

【図12F】

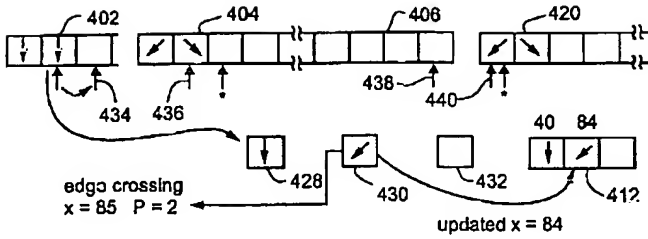


Fig. 12F

【図12G】

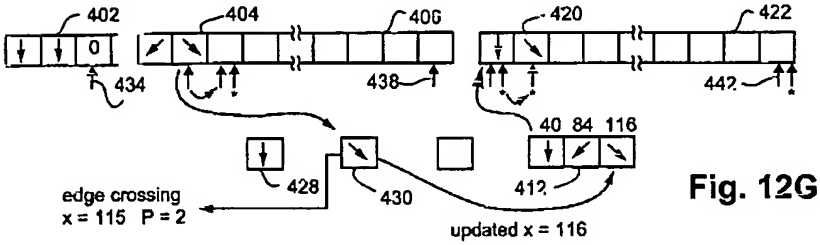


Fig. 12G

【図12H】

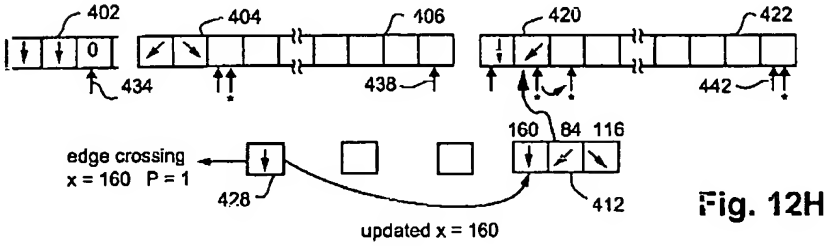


Fig. 12H

【図12I】

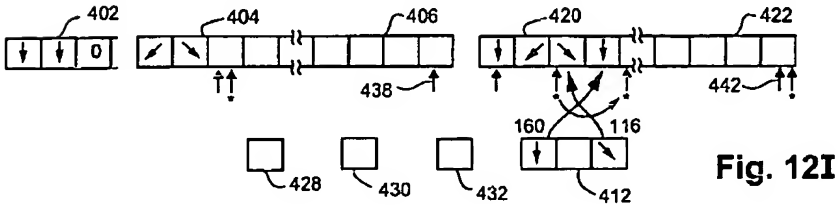


Fig. 12I

【図12J】

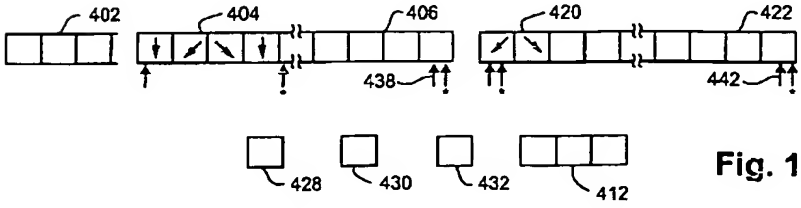


Fig. 12J

【図13B】

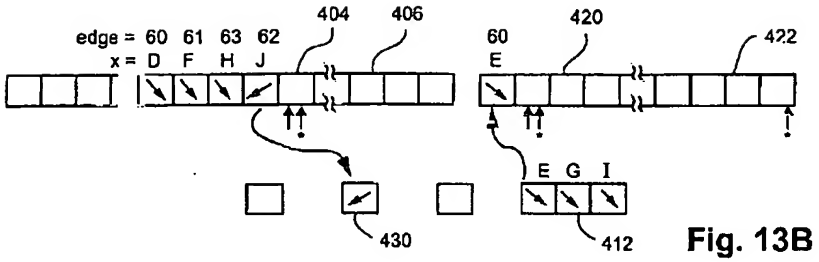


Fig. 13B

【図13C】

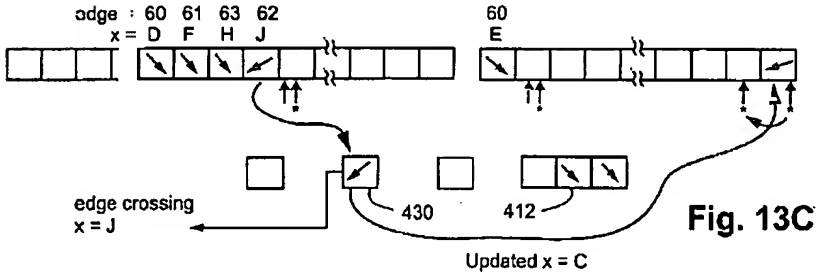


Fig. 13C

【図13D】

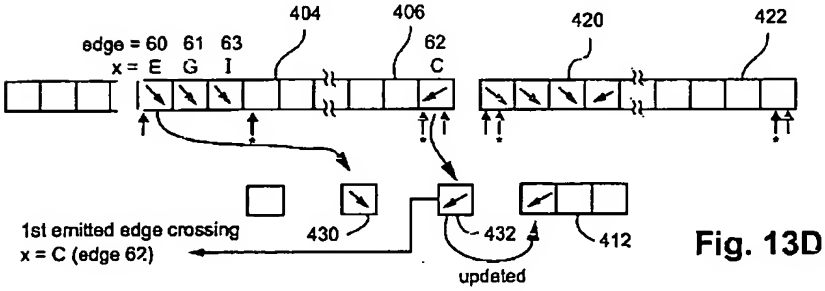


Fig. 13D

【図13E】

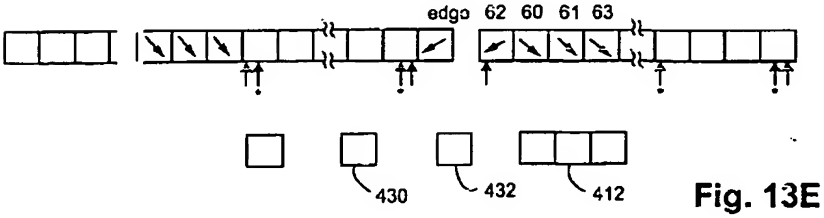
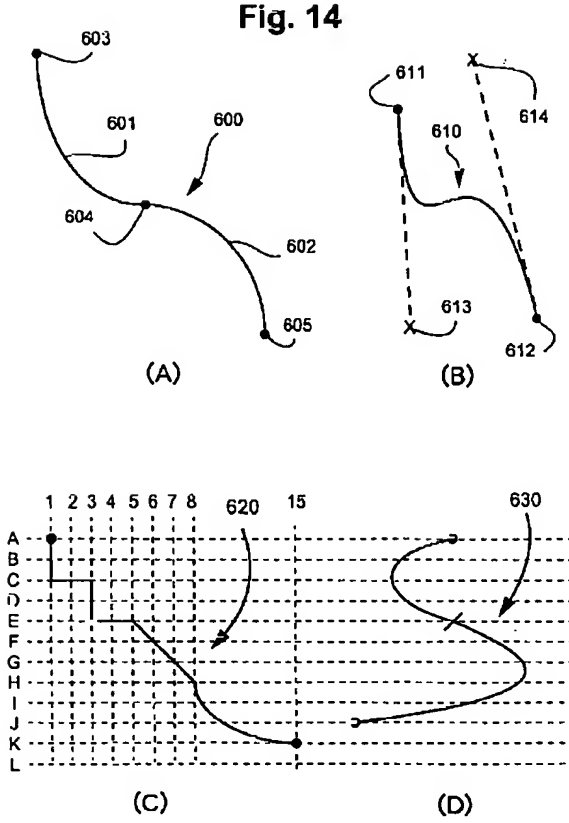


Fig. 13E

【図14】

Fig. 14



【図15】

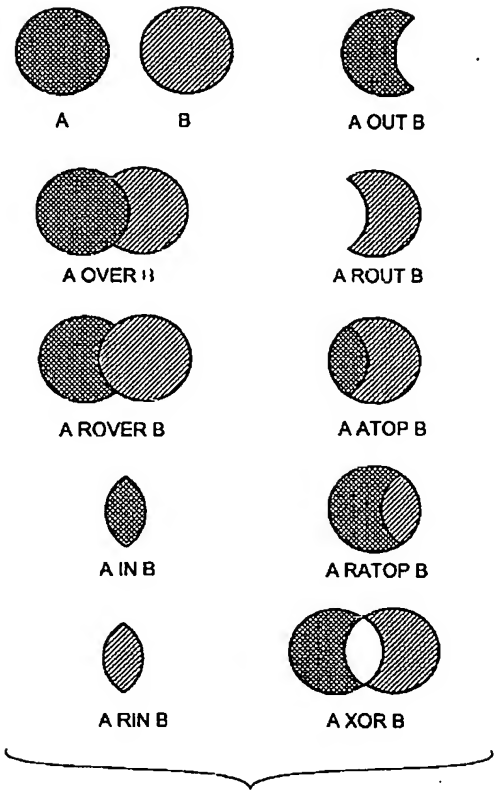


Fig. 15

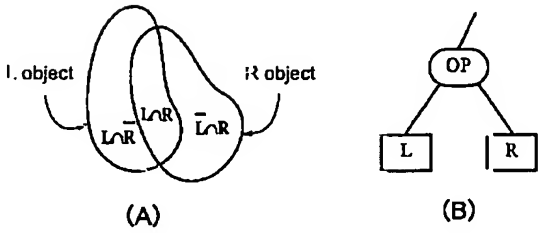
【図16】

Edge 84	Edge 92
X = 100	X = 40
NY = 70	NY = 70
DX = 1	DX = 0
DDX = 0	DDX = 0
P = 1	P = 0
DIR = (-)	DIR = (+)
ADD = (irrelevant in this example)	ADD = (irrelevant in this example)

Fig. 16

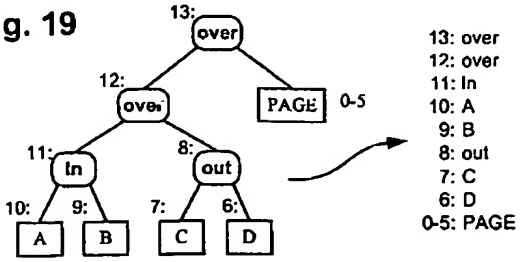
【図18】

Fig. 18



【図19】

Fig. 19



【図26】

L	T	T	T	L	L	I.	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	0	[illegible]	1	0	[illegible]	1
Generate R	0	0	0	0	0	0	0	[illegible]	0
Result	T	T	T	T	[illegible]	I.	T	O	O

Fig. 26

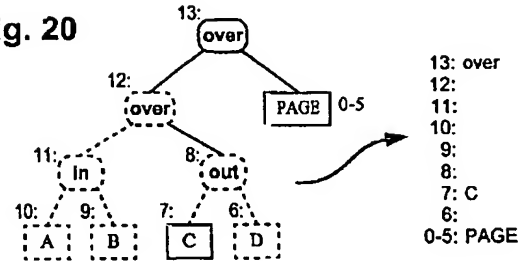
【図17】

Fig. 17

Raster operation code	Operation	Comment
0x00	r = 0	BLACKNESS
0x01	r = src & dest	SRCAND
0x02	r = src & ~dest	SRCERASE
0x03	r = src	SRCCOPY
0x04	r = ~src & dest	
0x05	r = dest	NOP
0x06	r = src ^ dest	SRCINVERT
0x07	r = src | dest	SRCPAINT
0x08	r = ~(src | dest)	NOTSRCERASE
0x09	r = ~(src ^ dest)	
0x0a	r = ~dest	DSTINVERT
0x0b	r = src | ~dest	
0x0c	r = ~src	NOTSRCCOPY
0x0d	r = ~src | dest	MERGEPAINT
0x0e	r = ~(src & dest)	
0x0f	r = 0xff	WHITENESS
0x10	r = min(src, dest)	
0x11	r = max(src, dest)	
0x12	r = clamp(src + dest)	
0x13	r = src	
0x14	r = clamp(src - dest)	
0x15	r = dest	
0x16	r = clamp(dest - src)	
0x17	r = clamp(src + dest) where dest is signed	
0x18	r = threshold (dest, src)	
0x19	r = threshold (src, dest)	
0x1a	r = ~dest	
0x1b	o = luminance (dest, src)	
0x1c	r = ~src	
0x1d	o = ckey(dest; src +/- o)	

【図20】

Fig. 20



【図27】

L	T	T	T	L	L	L	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	0		1	0		1
Generate R	0	0	0	0		1	0		1
Result	T	T	T	T		L	T		O

Fig. 27

【図29】

L	T	T	T	L	L	L	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	1		0	1		0
Generate R	0	0	0	0		0	0		0
Result	T	T	T	L		T	O		T

Fig. 29

【図21】

Index	L Active	R Active	L∩R̄ reqd	L̄∩R reqd	Leaf/ Operator Entry	Node Active	Parent	Node is L	Generate L	Generate R	L∩R op used	R Branch Index
13	0	1	1	1	over	1	?	0	0	1	0	5
12	0	0	1	1	over	0	13	1	0	0	0	8
11	0	0	0	0	in	0	12	1	0	0	0	9
10					leaf A	0	11	1				
9					leaf B	0	11	0				
8	0	0	1	0	out	0	12	0	0	0	0	6
7					leaf C	0	8	1				
6					leaf D	0	8	0				
PAGE												

Fig. 21

【図22】

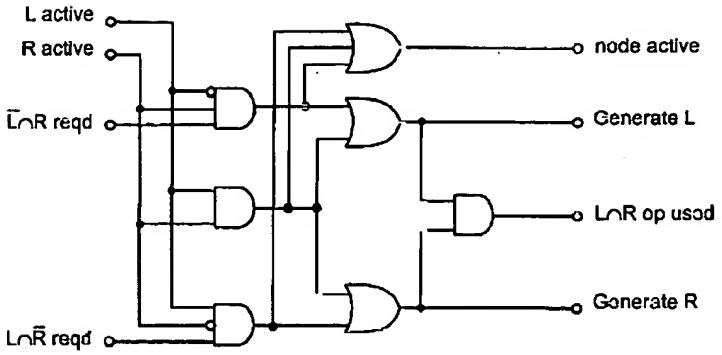


Fig. 22

【図23】

Index	L Active	R Active	L∩R̄ reqd	L̄∩R reqd	Leaf/ Operator Entry	Node Active	Parent	Node is L	Generate L	Generate R	L∩R op used	R Branch Index
13		1	1	1	over	1	?	0		1		5
12	0		1	1	over		13	1	0		0	8
11	0	0	0	0	In	0	12	1	0	0	0	9
10					leaf A	0	11	1				
9					leaf B	0	11	0				
8		0	1	0	out		12	0		0	0	6
7					leaf C		8	1				
6					leaf R	0	8	0				
PAGE												

Fig. 23

【図24】

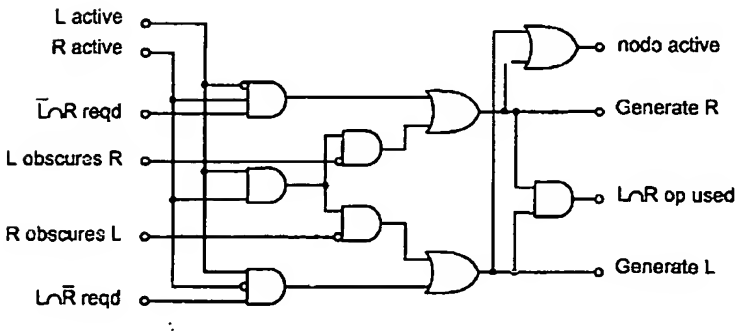


Fig. 24

【図25】

Index	L Active	R Active	L∩R̄ reqd	L̄∩R reqd	L obscures R	R obscures L	Leaf/ Operator Entry	Node Active	Parent	Node is L	Gene-rate L	Gene-rate R	L∩R op used	R Branch Index
			0	0	0	0	in							
			0	0	1	0	(CLIP IN)							
			1	0	0	0	out							
			1	0	1	1	(CLIP OUT							

Fig. 25

【図28】

L	T	T	T	L	L	L	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	1	[illegible]	1	1	[illegible]	1
Generate R	0	0	0	0	[illegible]	1	0	[illegible]	1
Result	T	T	T	L	[illegible]	T	O	[illegible]	T

Fig. 28

【図30】

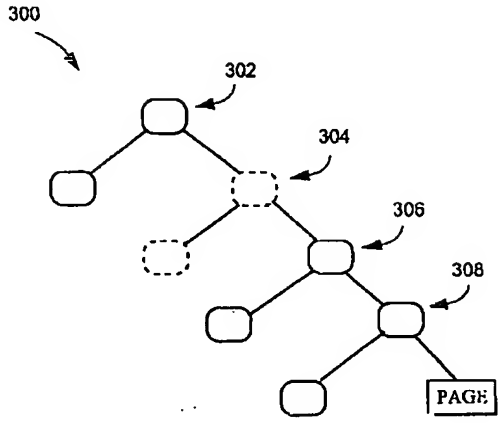


Fig. 30

【外国語明細書】

1. Title of the Invention

METHOD OF RENDERING GRAPHIC OBJECT BASED IMAGES AND APPARATUS THEREFOR

2. Claims

1. A method of generating instructions for a directed adjacency graph, said directed adjacency graph comprising one or more parent nodes and one or more leaf nodes, each of which said parent node representing an operator and having branches to respective descendent nodes, and each of which said leaf node representing a graphic object, said method comprising the steps of:

determining groups of one or more pixel locations;

determining, for each said group, a portion of said directed adjacency graph in accordance with activities of the operators, wherein the said portion of the directed adjacency graph is that portion which passes data up the directed adjacency graph; and

generating, for each said group, instructions for the determined portion of the directed adjacency graph, wherein operator instructions are generated for those operators of the determined portion of the directed adjacency graph having active branches and wherein leaf instructions are generated for those graphic objects which are active at said group of one or more pixel locations.

2. The method as claimed in claim 1, wherein said method comprises the step of generating a table for storing data concerning the activity of the operators and the activity of the branches of the parent nodes.

3. The method as claimed in claim 2, wherein said determining step comprises the sub-step of updating, for each said group, the said data stored in said table.

4. The method as claimed in claim 1, wherein said step of generating instructions comprises a sub-step of traversing said portion of the directed adjacency graph and generating instructions for the active operators and the leaf value instructions in said portion of the directed adjacency graph.

5. The method as claimed in claim 1, wherein said directed adjacency graph is an expression tree.

6. The method as claimed in claim 1, wherein said parent nodes represent binary operators.

7. A method of generating instructions for an expression tree, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said method comprising the steps of:

determining groups of one or more pixel locations;

determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive;

traversing, for each said group, said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; and

generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node.

8. A method as claimed in claim 7, wherein said tranversing step traverses said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously

traversed binary node are active and if the graphic object representing said descendent node on the right branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the left branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node, and wherein the right branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the left branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the right branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node.

9. A method of rendering an expression tree into a raster pixel image having a plurality of scanlines and a plurality of pixel locations on each said scanline, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said method comprising the steps of:

generating a table representative of said expression tree, wherein said table comprises a plurality of records corresponding to associated said binary nodes and leaf nodes, and each said record of a said associated binary node comprises a first field indicating whether a said left region is required for operation of the operator of said associated binary node, a second field indicating whether a right region is required for operation of the operator of said associated binary node, a third field capable of indicating whether a said left branch of said associated binary node is active, and a fourth field capable of indicating whether a said right branch of said associated binary node is active;

determining groups of one or more pixel locations;

determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive;

updating, for each said group, said third and fourth fields of said table for said determined active and inactive branches;

traversing, for each said group, said expression tree in accordance with said updated table wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node;

generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node; and

executing, for each said group, corresponding said generated instructions so as to render said image.

10. A method as claimed in claim 9, wherein said table further comprises for each said record of a said associated binary node a fifth field indicating whether the graphic object representing said descendent node on the right branch of said associated binary node obscures the graphic object representing said descendent node on the left branch of said associated binary node in the common region of the graphic objects representing said descendent nodes of said associated binary node, and a sixth field indicating whether the graphic object representing said descendent node on the left branch of said associated binary node obscures the graphic object representing said descendent node on the right branch of said associated binary node in the common region of the graphic objects representing said descendent nodes of said associated binary node; and said tranversing step traverses said expression tree in accordance with said updated table, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the right branch of said previously traversed binary node does not obscure the graphic object representing said

descendent node on the left branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node, and wherein the right branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the left branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the right branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node.

11. A method as claimed in claim 10, wherein said fifth and sixth fields are used to implement CLIP IN or a CLIP OUT operation.

12. Apparatus for generating instructions for a directed adjacency graph, said directed adjacency graph comprising one or more parent nodes and one or more leaf nodes, each of which said parent node representing an operator and having branches to respective descendent nodes, and each of which said leaf node representing a graphic object, said apparatus comprising:

means for determining groups of one or more pixel locations;

means for determining, for each said group, a portion of said directed adjacency graph in accordance with activities of the operators, wherein the said portion of the directed adjacency graph is that portion which passes data up the directed adjacency graph; and

means for generating, for each said group, instructions for the determined portion of the directed adjacency graph, wherein operator instructions are generated for those operators of the determined portion of the directed adjacency graph having active branches and wherein leaf instructions are generated for those graphic objects which are active at said group of one or more pixel locations.

13. Apparatus as claimed in claim 12, wherein said apparatus comprises means for generating a table for storing data concerning the activity of the operators and the activity of the branches of the parent nodes.

14. Apparatus as claimed in claim 13, wherein said determining means comprises means for updating, for each said group, the said data stored in said table.

15. Apparatus as claimed in claim 12, wherein said means for generating instructions comprises means for traversing said portion of the directed adjacency graph and means for generating instructions for active operators and leaf value instructions in said portion of the directed adjacency graph.

16. Apparatus as claimed in claim 12, wherein said directed adjacency graph is an expression tree.

17. Apparatus as claimed in claim 12, wherein said parent nodes represent binary operators.

18. Apparatus for generating instructions for an expression tree, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leat nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said apparatus comprising:

means for determining groups of one or more pixel locations;

means for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive;

means for traversing, for each said group, said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; and

means for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node.

19. Apparatus as claimed in claim 18, wherein said tranversing means traverses said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the right branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the left branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node, and wherein the right branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the left branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the right branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node.

20. Apparatus for rendering an expression tree into a raster pixel image having a plurality of scanlines and a plurality of pixel locations on each said scanline, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said apparatus comprising:

means for generating a table representative of said expression tree, wherein said table comprises a plurality of records corresponding to associated said binary nodes and leaf nodes, and each said record of a said associated binary node comprises a first field indicating whether a said left region is required for operation of the operator of said associated binary node, a second field indicating whether a right region is required for

operation of the operator of said associated binary node, a third field capable of indicating whether a said left branch of said associated binary node is active, and a fourth field capable of indicating whether a said right branch of said associated binary node is active;

means for determining groups of one or more pixel locations;

means for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive;

means for updating, for each said group, said third and fourth fields of said table for said determined active and inactive branches;

means for traversing, for each said group, said expression tree in accordance with said updated table wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node;

means for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node; and

means for executing, for each said group, corresponding said generated instructions so as to render said image.

21. Apparatus as claimed in claim 20, wherein said table further comprises for each said record of a said associated binary node a fifth field indicating whether the graphic object representing said descendent node on the right branch of said associated binary node obscures the graphic object representing said descendent node on the left branch of said associated binary node in the common region of the graphic objects representing said descendent nodes of said associated binary node, and a sixth field indicating whether the graphic object representing said descendent node on the left branch of said associated binary node obscures the graphic object representing said descendent node on the right branch of said associated binary node in the common region of the graphic objects

representing said descendent nodes of said associated binary node; and said tranversing means traverses said expression tree in accordance with said updated table, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the right branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the left branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node, and wherein the right branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the left branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the right branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node.

22. Apparatus as claimed in claim 21, wherein said fifth and sixth fields are used to implement CLIP IN or a CLIP OUT operation.

23. A computer readable medium comprising a computer program for generating instructions for a directed adjacency graph, said directed adjacency graph comprising one or more parent nodes and one or more leaf nodes, each of which said parent node representing an operator and having branches to respective descendent nodes, and each of which said leaf node representing a graphic object, said computer program comprising:

code for determining groups of one or more pixel locations;

code for determining, for each said group, a portion of said directed adjacency graph in accordance with activities of the operators, wherein the said portion of the directed adjacency graph is that portion which passes data up the directed adjacency graph; and

code for generating, for each said group, instructions for the determined portion of the directed adjacency graph, wherein operator instructions are generated for those operators of the determined portion of the directed adjacency graph having active branches and wherein leaf instructions are generated for those graphic objects which are active at said group of one or more pixel locations.

24. A computer readable medium as claimed in claim 23, wherein said computer program comprises code for generating a table for storing data concerning the activity of the operators and the activity of the branches of the parent nodes.

25. A computer readable medium as claimed in claim 24, wherein said determining code comprises code for updating, for each said group, the said data stored in said table.

26. A computer readable medium as claimed in claim 23, wherein said code for generating instructions comprises code for traversing said portion of the directed adjacency graph and code for generating instructions for active operators and leaf value instructions in said portion of the directed adjacency graph.

27. A computer readable medium as claimed in claim 23, wherein said directed adjacency graph is an expression tree.

28. A computer readable medium as claimed in claim 23, wherein said parent nodes represent binary operators.

29. A computer readable medium comprising a computer program for generating instructions for an expression tree, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said computer program comprising:

code for determining groups of one or more pixel locations;

code for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive;

code for traversing, for each said group, said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed

binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; and

code for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node.

30. A computer readable medium as claimed in claim 29, wherein said tranversing code traverses said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the right branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the left branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node, and wherein the right branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the left branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the right branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node.

31. A computer readable medium comprising a computer program for rendering an expression tree into a raster pixel image having a plurality of scanlines and a plurality of pixel locations on each said scanline, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic

objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said computer program comprising:

code for generating a table representative of said expression tree, wherein said table comprises a plurality of records corresponding to associated said binary nodes and leaf nodes, and each said record of a said associated binary node comprises a first field indicating whether a said left region is required for operation of the operator of said associated binary node, a second field indicating whether a right region is required for operation of the operator of said associated binary node, a third field capable of indicating whether a said left branch of said associated binary node is active, and a fourth field capable of indicating whether a said right branch of said associated binary node is active;

code for determining groups of one or more pixel locations;

code for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive;

code for updating, for each said group, said third and fourth fields of said table for said determined active and inactive branches;

code for traversing, for each said group, said expression tree in accordance with said updated table wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node;

code for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node; and

code for executing, for each said group, corresponding said generated instructions so as to render said image.

32. A computer readable medium as claimed in claim 31, wherein said table further comprises for each said record of a said associated binary node a fifth field indicating

whether the graphic object representing said descendent node on the right branch of said associated binary node obscures the graphic object representing said descendent node on the left branch of said associated binary node in the common region of the graphic objects representing said descendent nodes of said associated binary node, and a sixth field indicating whether the graphic object representing said descendent node on the left branch of said associated binary node obscures the graphic object representing said descendent node on the right branch of said associated binary node in the common region of the graphic objects representing said descendent nodes of said associated binary node; and said tranversing means traverses said expression tree in accordance with said updated table, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the right branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the left branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node, and wherein the right branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active and if the graphic object representing said descendent node on the left branch of said previously traversed binary node does not obscure the graphic object representing said descendent node on the right branch of said previously traversed binary node in the common region of the graphic objects representing said descendent nodes of said previously traversed binary node.

33. A computer program as claimed in claim 32, wherein said fifth and sixth fields are used to implement CLIP IN or a CLIP OUT operation.

3. Detailed Description of the Invention

Copyright Notice

Technical Field of the Invention

The present invention relates generally to rendering graphic object based images. In particular, the present invention relates to a method and apparatus for generating instructions for an expression tree. The invention also relates to a method and apparatus for rendering an expression tree into a raster pixel image. The invention also relates to a computer readable medium comprising a computer program for implementing any of the aforesaid methods.

Background

Most object based graphics systems utilise a frame store or page buffer to hold a pixel-based image of the page or screen. Typically, the outlines of the graphic objects are calculated, filled and written into the frame store. For two-dimensional graphics, objects which appear in front of other objects are simply written into the frame store after the background objects, thereby replacing the background on a pixel-by-pixel basis. This is commonly known in the art as the "Painter's algorithm". Objects are considered in priority order, from the rearmost object to the foremost object, and, typically, each object is rasterised in scan line order and pixels are written to the frame store in sequential runs along each scan line.

There are essentially two problems with this technique. The first is that it requires fast random access to all the pixels in the frame store. This is because each new object considered could affect any pixel in the frame-store. For this reason, the frame store is normally kept in semiconductor random access memory (RAM). For high resolution colour printers the amount of RAM required is very large, typically in excess of 100 MBytes, which is costly and difficult to operate at high speed. The second problem is that many pixels, which are painted (rendered), are over-painted (re-rendered) by later objects. Painting the pixels with the earlier objects was a waste of time.

One method for overcoming the large frame-store problem is the use of "banding". When banding is used, only part of the frame-store exists in memory at any one time. All of the objects to be drawn are retained in a "display list". The whole image is rendered as

above, but pixel painting (rendering) operations that try to paint (render) outside the fraction of the frame-store which exists are "clipped" out. After all the objects have been drawn, the fractional part of the frame-store is sent to the printer (or some other location) and another fraction of the frame-store is selected and the process repeated. There are penalties with this technique. For example, the objects being drawn must be considered and re-considered many times - once for each band. As the number of bands increases, so too does the repetitious examination of objects requiring rendering. The technique of banding does not solve the problem of the cost of over-painting.

Some other graphic systems consider the image in scan line order. Again, all the objects to be drawn are retained in a display list. On each scan line the objects which intersect that scan line are then considered in priority order and for each object, spans of pixels between object edge intersection points are set in a line store. This technique also overcomes the large frame store problem, but still suffers from the over-paint problem.

There are other techniques, which overcome both the large frame-store problem and the over-painting problem. In one such technique, each scan line is produced in turn. Again, all the objects to be drawn are retained in a display list. On each scan line, the edges of objects which intersect that scan line are held in order of increasing coordinate of intersection with the scan line. These points of intersection, or edge crossings, are considered in turn and used to toggle an array of active fields. There is one active field for each object priority that is of interest on the scan line. Between each pair of edges considered, the colour data for each pixel which lies between the first edge and the next edge is generated by using a priority encoder on the active flags to determine which priority is topmost, and using the colour associated with that priority for the pixels of the span between the two edges. In preparation for the next scan line, the coordinate of intersection of each edge is updated in accordance with the nature of each edge. Adjacent edges, which become mis-sorted as a result of this update are swapped. New edges are also merged into the list of edges.

This technique has the significant advantages that there is no frame store or line store, there is no over painting, and the object priorities are dealt with in constant order time, rather than order N time (where N is the number of priorities).

However, these priority encoders lack sufficient flexibility to cope with a tree based graphical expression. In particular, those expressions which include OUT and ATOP Porter and Duff compositing operators as described in "Compositing Digital Images", Porter, T; Duff, T; Computer Graphics, Vol. 18 No. 3 (1984) pp. 253-259. The

problems arise because the compositing operation depends on which of the graphic objects composited by the operation are active at a given pixel location. One solution to this problem is to use a complex arrangement of clipping objects, which requires a lot of extra edge processing and requires a large number of levels for the clipping objects.

Summary of the Invention

It is an object of the present invention to substantially overcome, or at least ameliorate, one or more disadvantages of existing arrangements.

According to a first aspect of the invention, there is provided a method of generating instructions for a directed adjacency graph, said directed adjacency graph comprising one or more parent nodes and one or more leaf nodes, each of which said parent node representing an operator and having branches to respective descendent nodes, and each of which said leaf node representing a graphic object, said method comprising the steps of: determining groups of one or more pixel locations; determining, for each said group, a portion of said directed adjacency graph in accordance with activities of the operators, wherein the said portion of the directed adjacency graph is that portion which passes data up the directed adjacency graph; and generating, for each said group, instructions for the determined portion of the directed adjacency graph, wherein operator instructions are generated for those operators of the determined portion of the directed adjacency graph having active branches and wherein leaf instructions are generated for those graphic objects which are active at said group of one or more pixel locations.

According to a second aspect of the invention, there is provided a method of generating instructions for an expression tree, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said method comprising the steps of: determining groups of one or more pixel locations; determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive; traversing, for each said group, said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node

and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; and generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node.

According to a third aspect of the invention, there is provided a method of rendering an expression tree into a raster pixel image having a plurality of scanlines and a plurality of pixel locations on each said scanline, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said method comprising the steps of: generating a table representative of said expression tree, wherein said table comprises a plurality of records corresponding to associated said binary nodes and leaf nodes, and each said record of a said associated binary node comprises a first field indicating whether a said left region is required for operation of the operator of said associated binary node, a second field indicating whether a right region is required for operation of the operator of said associated binary node, a third field capable of indicating whether a said left branch of said associated binary node is active, and a fourth field capable of indicating whether a said right branch of said associated binary node is active; determining groups of one or more pixel locations; determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive; updating, for each said group, said third and fourth fields of said table for said determined active and inactive branches; traversing, for each said group, said expression tree in accordance with said updated table wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node,

and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node; and executing, for each said group, corresponding said generated instructions so as to render said image.

According to a fourth aspect of the invention, there is provided apparatus for generating instructions for a directed adjacency graph, said directed adjacency graph comprising one or more parent nodes and one or more leaf nodes, each of which said parent node representing an operator and having branches to respective descendent nodes, and each of which said leaf node representing a graphic object, said apparatus comprising: means for determining groups of one or more pixel locations; means for determining, for each said group, a portion of said directed adjacency graph in accordance with activities of the operators, wherein the said portion of the directed adjacency graph is that portion which passes data up the directed adjacency graph; and means for generating, for each said group, instructions for the determined portion of the directed adjacency graph, wherein operator instructions are generated for those operators of the determined portion of the directed adjacency graph having active branches and wherein leaf instructions are generated for those graphic objects which are active at said group of one or more pixel locations.

According to a fifth aspect of the invention, there is provided apparatus for generating instructions for an expression tree, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said apparatus comprising: means for determining groups of one or more pixel locations; means for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive; means for traversing, for each said group, said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent

node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; and means for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node.

According to a sixth aspect of the invention, there is provided apparatus for rendering an expression tree into a raster pixel image having a plurality of scanlines and a plurality of pixel locations on each said scanline, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said apparatus comprising: means for generating a table representative of said expression tree, wherein said table comprises a plurality of records corresponding to associated said binary nodes and leaf nodes, and each said record of a said associated binary node comprises a first field indicating whether a said left region is required for operation of the operator of said associated binary node, a second field indicating whether a right region is required for operation of the operator of said associated binary node, a third field capable of indicating whether a said left branch of said associated binary node is active, and a fourth field capable of indicating whether a said right branch of said associated binary node is active; means for determining groups of one or more pixel locations; means for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive; means for updating, for each said group, said third and fourth fields of said table for said determined active and inactive branches; means for traversing, for each said group, said expression tree in accordance with said updated table wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left

node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; means for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node; and means for executing, for each said group, corresponding said generated instructions so as to render said image.

According to a seventh aspect of the invention, there is provided a computer readable medium comprising a computer program for generating instructions for a directed adjacency graph, said directed adjacency graph comprising one or more parent nodes and one or more leaf nodes, each of which said parent node representing an operator and having branches to respective descendent nodes, and each of which said leaf node representing a graphic object, said computer program comprising: code for determining groups of one or more pixel locations; code for determining, for each said group, a portion of said directed adjacency graph in accordance with activities of the operators, wherein the said portion of the directed adjacency graph is that portion which passes data up the directed adjacency graph; and code for generating, for each said group, instructions for the determined portion of the directed adjacency graph, wherein operator instructions are generated for those operators of the determined portion of the directed adjacency graph having active branches and wherein leaf instructions are generated for those graphic objects which are active at said group of one or more pixel locations.

According to an eighth aspect of the invention, there is provided a computer readable medium comprising a computer program for generating instructions for an expression tree, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said computer program comprising: code for determining groups of one

or more pixel locations; code for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive; code for traversing, for each said group, said expression tree, wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; and code for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node.

According to a ninth aspect of the invention, there is provided a computer readable medium comprising a computer program for rendering an expression tree into a raster pixel image having a plurality of scanlines and a plurality of pixel locations on each said scanline, said expression tree having a plurality of nodes comprising one or more binary nodes and a plurality of leaf nodes, wherein each said binary node having a left branch to a descendent said node and a right branch to a descendent said node and representing a binary operation on said two descendant nodes, and wherein each said node represents a graphic object, with one or more said graphic objects overlapping, each said overlapping graphics objects comprising a left node region, a common region, and a right node region, said computer program comprising: code for generating a table representative of said expression tree, wherein said table comprises a plurality of records corresponding to associated said binary nodes and leaf nodes, and each said record of a said associated binary node comprises a first field indicating whether a said left region is required for operation of the operator of said associated binary node, a second field indicating whether a right region is required for operation of the operator of said associated binary node, a third field capable of indicating whether a said left branch of said associated binary node is active, and a fourth field capable of indicating whether a said right branch of said associated binary node is active; code for determining groups of one or more pixel locations; code for determining, for each said group, whether the left and right branches of said one or more binary nodes are active or inactive; code for updating, for each said

group, said third and fourth fields of said table for said determined active and inactive branches; code for traversing, for each said group, said expression tree in accordance with said updated table wherein the left branch of any previously traversed said binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said left node region is required for the binary operation of said previously traversed binary node and the left branch is active and the right branch is inactive of said previously traversed binary node, and wherein a right branch of any previously traversed binary node is traversed to its said descendent node if the right and left branches of said previously traversed binary node are active or if a said right node region is required for the binary operation of said previously traversed binary node and the right branch is active and the left branch is inactive of said previously traversed binary node; code for generating, for each said group, operator instructions for any said traversed binary node having active said right and left branches, and leaf value instructions for any traversed leaf node; and code for executing, for each said group, corresponding said generated instructions so as to render said image.

Detailed Description including Best Mode

Where reference is made in any one or more of the accompanying drawings to steps and/or features, which have the same reference numerals, those steps and/or features have for the purposes of this description the same function(s) or operation(s), unless the contrary intention appears.

Fig. 1 illustrates schematically a computer system 1 configured for rendering and presentation of computer graphic object images. The system includes a host processor 2 associated with system random access memory (RAM) 3, which may include a non-volatile hard disk drive or similar device 5 and volatile, semiconductor RAM 4. The system 1 also includes a system read-only memory (ROM) 6 typically founded upon semiconductor ROM 7 and which in many cases may be supplemented by compact disk devices (CD ROM) 8. The system 1 may also incorporate some means 10 for displaying images, such as a video display unit (VDU) or a printer, both of which operate in raster fashion.

The above-described components of the system 1 are interconnected via a bus system 9 and are operable in a normal operating mode of computer systems well known in the art, such as IBM PC/AT type personal computers and arrangements evolved therefrom, Sun Sparcstations and the like.

Also seen in Fig. 1, a pixel sequential rendering apparatus 20 connects to the bus 9, and in the preferred embodiment is configured for the sequential rendering of pixel-based images derived from graphic object-based descriptions supplied with instructions and data from the system 1 via the bus 9. The apparatus 20 may utilise the system RAM 3 for the rendering of object descriptions although preferably the rendering apparatus 20 may have associated therewith a dedicated rendering store arrangement 30, typically formed of semiconductor RAM.

The general principles of the invention have application in generating instructions for directed adjacency graphs, and specifically expression trees. This is realised in the preferred embodiment in an activity determination and instruction generation module 500 (Fig. 5) of the pixel sequential rendering apparatus 20. This module is described in more detail in the section herein entitled *"3.0 Activity Determination and Instruction Generation Module"*.

Referring now to Fig. 2, the overall functional data flow diagram of the preferred embodiment is shown. The functional flow diagram of Fig. 2 commences with an object graphic description 11 which is used to describe those parameters of graphic objects in a fashion appropriate to be generated by the host processor 2 and/or, where appropriate, stored within the system RAM 3 or derived from the system ROM 6, and which may be interpreted by the pixel sequential rendering apparatus 20 to render therefrom pixel-based images. For example, the object graphic description 11 may incorporate objects with edges in a number of formats including straight edges (simple vectors) that traverse from one point on the display to another, or an orthogonal edge format where a two-dimensional object is defined by a plurality of edges including orthogonal lines. Further formats, where objects are defined by continuous curves, are also appropriate and these can include quadratic polynomial fragments where a single curve may be described by a number of parameters which enable a quadratic based curve to be rendered in a single output space without the need to perform multiplications. Further data formats such as cubic splines and the like may also be used. An object may contain a mixture of many different edge types. Typically, common to all formats are identifiers for the start and end of each line (whether straight or curved) and typically, these are identified by a scan line number thus defining a specific output space in which the curve may be rendered.

For example, Fig. 14A shows a prior art edge description of an edge 600 that is required to be divided into two segments 601 and 602 in order for the segments to be adequately described and rendered. This arises because the prior art edge description, whilst being simply calculated through a quadratic expression, could not accommodate an inflexion point 604. Thus the edge 600 was dealt with as two separate edges having end points 603 and 604, and 604 and 605 respectively. Fig. 14B shows a cubic spline 610 which is described by end points 611 and 612, and control points 613 and 614. This format requires calculation of a cubic polynomial for render purposes and thus is expensive of computational time.

Figs. 14C and 14D show examples of edges applicable to the preferred embodiment. In the preferred embodiment, a edge is considered as a single entity and if necessary, is partitioned to delineate sections of the edge that may be described in different formats, a specific goal of which is to ensure a minimum level of complexity for the description of each section.

In Fig. 14C, a single edge 620 is illustrated spanning between scan lines A and M. An edge is described by a number of parameters including start_x , start_y , one or more

segment descriptions which include an address that points to the next segment in the edge, and a finish segment used to terminate the edge. According to the preferred embodiment, the edge 620 may be described as having three step segments, a vector segment, and a quadratic segment. A step segment is simply defined as having an x-step value and a y-step value. For the three step segments illustrated, the segment descriptions are [0,2], [+2,2], and [+2,0]. Note that the x-step value is signed thereby indicating the direction of the step, whilst the y-step value is unsigned as such is always in a raster scan direction of increasing scan line value. The next segment is a vector segment which typically requires parameters start_x, start_y, finish_y and slope (DX). In this example, because the vector segment is an intermediate segment of the edge 620, the start_x and start_y may be omitted because such arise from the preceding segment(s). The slope value (DX) is signed and is added to the x-value of a preceding scan line to give the x-value of the current scan line, and in the illustrated case, DX = +1. The next segment is a quadratic segment which has a structure corresponding to that of the vector segment, but also a second order value (DDX) which is also signed and is added to DX to alter the slope of the segment.

Fig. 14D shows an example of a cubic curve according the preferred embodiment which includes a description corresponding to the quadratic segment save for the addition of a signed third-order value (DDDX), which is added to DDX to vary the rate of change of slope of the segment. Many other orders may also be implemented.

It will be apparent from the above that the ability to handle plural data formats describing edge segments allows for simplification of edge descriptions and evaluation, without reliance on complex and computationally expensive mathematical operations. In contrast, in the prior art system of Fig. 14A, all edges, whether, orthogonal, vector or quadratic, were required to be described by the quadratic form.

The operation of the preferred embodiment will be described with reference to the simple example of rendering an image 78 shown in Fig. 8. The image 78 is seen to include two graphical objects, in particular, a partly transparent blue-coloured triangle 80 rendered on top of and thereby partly obscuring an opaque red coloured rectangle 90. As seen, the rectangle 90 includes side edges 92, 94, 96 and 98 defined between various pixel positions (X) and scan line positions (Y). Because the edges 96 and 98 are formed upon the scan lines (and thus parallel therewith), the actual object description of the rectangle 90 can be based solely upon the side edges 92 and 94, such as seen in Fig. 9A. In this connection, edge 92 commences at pixel location (40,35) and extends in a raster

direction down the screen to terminate at pixel position (40,105). Similarly, the edge 94 extends from pixel position (160,35) to position (160,105). The horizontal portions of the rectangular graphic object 90 may be obtained merely by scanning from the edge 92 to the edge 94 in a rasterised fashion.

The blue triangular object 80 however is defined by three object edges 82, 84 and 86, each seen as vectors that define the vertices of the triangle. Edges 82 and 84 are seen to commence at pixel location (100,20) and extend respectively to pixel locations (170,90) and (30,90). Edge 86 extends between those two pixel locations in a traditional rasterised direction of left to right. In this specific example because the edge 86 is horizontal like the edges 96 and 98 mentioned above, is it not essential that the edge 86 be defined, since the edge 86 is characterised by the related endpoints of the edges 82 and 84. In addition to the starting and ending pixel locations used to describe the edges 82 and 84, each of these edges will have associated therewith the slope value in this case +1 and -1 respectively.

Fig. 10 shows the manner in which the rectangle 90 is rendered, this commencing on scan line 35 and how the edges 82 and 84 intersect the scan line 35. It will be apparent from Fig. 10 that the rasterisation of the image 78 requires resolution of the two objects 90 and 80 in such a fashion that the object having the higher priority level is rendered "above" that with a lower priority level. This is seen from Fig. 11 which represents an edge list record used for the rendering of the image 78. The record of Fig. 11 includes two entries, one for each of the objects, and which are arranged at a scan line value corresponding to the start, in a raster rendering order, of the respective object. It will be seen from Fig. 11 that the edge records each have an associated priority level of the object and further detail regarding the nature of the edge being described (eg. colour, slope, etc.)

Returning to Fig. 2, having identified the data necessary to describe the graphic objects to the rendered, the graphic systems 1 then performs a display list generation step 12.

The display list generation 12 is preferably implemented as a software module executing on the host processor 2 with attached ROM 6 and RAM 3. The display list generation 12 converts an object graphics description, expressed in any one or more of the well known graphic description languages, graphic library calls, or any other application specific format, into a display list. The display list is typically written into a display list store 13, generally formed within the RAM 4 but which may alternatively be formed

within the rendering stores 30. As seen in Fig. 3, the display list store 13 can include a number of components, one being an instruction stream 14, another being edge information 15 and where appropriate, raster image pixel data 16.

The instruction stream 14 includes code interpretable as instructions to be read by the pixel sequential rendering apparatus 20 to render the specific graphic objects desired in any specific image. For the example of the image shown in Fig. 8, the instruction stream 14 could be of the form of:

(1) render (nothing) to scan line 20;

(2) at scan line 20 add two blue edges 82 and 84;

(3) render to scan line 35;

(4) at scan line 35 add two red edges 92 and 94;

(5) render to completion.

Similarly, the edge information 15 for the example of Fig. 8 may include the following:

edge 84 commences at pixel position 100, edge 82 commences at pixel position 100;

edge 92 commences at pixel position 40, edge 94 commences at pixel position 160;

edge 84 runs for 70 scan lines, edge 82 runs for 70 scan lines;

edge 84 has slope = -1, edge 84 has slope = +1;

edge 92 has slope = 0 edge 94 has slope = 0; and

edges 92 and 94 each run for 70 scan lines.

It will be appreciated from the above example of the instruction stream 14 and edge information 15, and the manner in which each are expressed, that in the image 78 of Fig. 8, the pixel position (X) and the scan line value (Y) define a single output space in which the image 78 is rendered. Other output space configurations however can be realised using the principles of the present disclosure.

Fig. 8 includes no raster image pixel data and hence none need be stored in the store portion 16 of the display list 13, although this feature will be described later.

The display list store 13 is read by a pixel sequential rendering apparatus 20, which is typically implemented as an integrated circuit. The pixel sequential rendering apparatus 20 converts the display list into a stream of raster pixels which can be forwarded to another device, for example, a printer, a display, or a memory store.

Although the preferred embodiment describes the pixel sequential rendering apparatus 20 as an integrated circuit, it may be implemented as an equivalent software module executable on a general purpose processing unit, such as the host processor 2. The software module may form part of a computer program product which may be delivered to a user via a computer readable medium, such as a disk device or computer network.

Fig. 3 shows the configuration of the pixel sequential rendering apparatus 20, the display list store 13 and the temporary rendering stores 30. The processing stages 22 of the pixel-sequential render apparatus 20 include:

an instruction executor 300 (which is described in more detail in the section herein entitled *"1.0 Instruction Executor"*);

an edge processing module 400 (which is described in more detail in the section herein entitled "*2.0 Edge Processing Module*");

an activity determination and instruction generation module 500 (which is described in more detail in the section herein entitled "*3.0 Activity Determination and Instruction Generation Module*");

a fill colour determination module 600 (which is described in more detail in the section herein entitled "*4.0 Fill Colour Determination Module*");

a pixel compositing module 700 (which is described in more detail in the section herein entitled "*5.0 Pixel Compositing Module*"); and

a pixel output module 800 (which is described in more detail in the section herein entitled "*6.0 Pixel Output Module*");.

The processing operations use the temporary stores 30 which as noted above, may share the same device (eg. magnetic disk or semiconductor RAM) as the display list store 13, or may be implemented as individual stores for reasons of speed optimisation. The edge processing module 400 uses an edge record store 32 to hold edge information which is carried forward from scan-line to scan-line. The activity determination and instruction generation module 500 uses a level activation table 34 to hold information about operator performance, and the current state of the activity of each region with respect to edge crossings while a scan-line is being rendered. The fill colour determination module 600 uses a fill data table 36 to hold information required to determine the fill colour of a particular priority at a particular position. The pixel compositing module 700 uses a pixel compositing stack 38 to hold intermediate results

during the determination of an output pixel that requires the colours from multiple priorities to determine its value.

The display list store 13 and the other stores 32-38 detailed above may be implemented in RAM or any other data storage technology.

The processing steps shown in the embodiment of Fig. 3 take the form of a processing pipeline 22. In this case, the modules of the pipeline may execute simultaneously on different portions of image data in parallel, with messages passed between them as described below. In another embodiment, each message described below may take the form of a synchronous transfer of control to a downstream module, with upstream processing suspended until the downstream module completes the processing of the message.

1.0 Instruction Executor

The instruction executor 300 reads and processes instructions from the instruction stream 14 and formats the instructions into messages that are transferred via an output 398 to the other modules 400, 500, 600 and 700 within the pipeline 22. In the preferred embodiment, the instruction stream 14 may include the instructions:

LOAD_PRIORITY_PROPERTIES: This instruction is associated with data to be loaded into the level activation table 34, and an address in that table to which the data is to be loaded. When this instruction is encountered by the instruction executor 300, the instruction executor 300 issues a message for the storage of the data in the specified location of the level activation table 34. This may be accomplished by formatting a message containing this data and passing it down the processing pipeline 22 to the activity determination and instruction generation module 500 which performs the store operation.

LOAD_FILL_DATA: This instruction is associated with data to be loaded into the fill data table 36, and an address in that table to which the data is to be loaded. When this instruction is encountered by the instruction executor 300, the instruction executor 300 issues a message for the storage of the data at the specified address of the fill data table 36. This may be accomplished by formatting a message containing this data and passing it down the processing pipeline 22 to the fill data determination module which performs the store operation.

LOAD_NEW_EDGES_AND_RENDER: This instruction is associated with an address in the display list store 13 of new edges 15 which are to be introduced into the rendering process when a next scan line is rendered. When this instruction is encountered

by the instruction executor, the instruction executor 300 formats a message containing this data and passes it to the edge processing module 400. The edge processing module 400 stores the address of the new edges in the edge record store 32. The edges at the specified address are sorted on their initial scan line intersection coordinate before the next scan line is rendered. In one embodiment, the edges are sorted by the display list generation process 12. In another embodiment, the edges are sorted by the pixel-sequential rendering apparatus 20.

SET_SCAN LINE_LENGTH: This instruction is associated with a number of pixels which are to be produced in each rendered scan line. When this instruction is encountered by the instruction executor 300, the instruction executor 300 passes the value to the edge processing module 400 and the pixel compositing module 700.

SET_OPACITY _MODE: This instruction is associated with a flag which indicates whether pixel compositing operations will use an opacity channel (also known in the art as an alpha channel). When this instruction is encountered by the instruction executor 300, the instruction executor 300 passes the flag value in the pixel compositing module 700.

The instruction executor 300 is typically formed by a microcode state machine which maps instructions and decodes them into pipeline operations for passing to the various modules. A corresponding software process may alternatively be used.

2.0 Edge Processing Module

The operation of the edge processing module 400 during a scan line render operation will now be described with reference to Fig. 4. The initial conditions for the rendering of a scan line is the availability of three lists of edge records. Any or all of these lists may be empty. These lists are a new edge list 402, obtained from the edge information 15 and which contains new edges as set by the LOAD_NEW_EDGES_AND_RENDER instruction, a main edge list 404 which contains edge records carried forward from the previous scan line, and a spill edge list 406 which also contains edge records carried forward from the previous scan line. Each edge record may include:

(i) a current scan line intersection coordinate (referred to here as the X coordinate),

(ii) a count (referred to herein as NY) of how many scan lines a current segment of this edge will last for (in some embodiments this may be represented as a Y limit),

(iii) a value to be added to the X coordinate of this edge record after each scan line (referred to here as the DX),

(iv) a value to be added to the DX of this edge record after each scan line (referred to here as the DDX),

(v) one or more priority numbers (P),

(vi) a direction (DIR) flag which indicates whether the edge crosses scan lines in an upward (+) or a downward (-) manner, and

(vii) an address (ADD) of a next edge segment in the list.

Such a format accommodates vectors, orthogonally arranged edges and quadratic curves. The addition of further parameters, DDDX for example, may allow such an arrangement to accommodate cubic curves. In some applications, such as cubic Bezier spline, a 6-order polynomial (ie: up to DDDDDDX) may be required.

For the example of the edges 84 and 94 of Fig. 8, the corresponding edge records at scan line 20 could read as follows in the Table as shown in Fig. 16 of the drawings:

In this description, coordinates which step from pixel to pixel along a scan line being generated by the rendering process will be referred to as X coordinates, and coordinates which step from scan line to scan line will be referred to as Y coordinates. Preferably, each edge list contains zero or more records placed contiguously in memory. Other storage arrangements, including the use of pointer chains, are also possible. The records in each of the three lists 402, 404 and 406 are arranged in order of scan line intersection (X) coordinate. This is typically obtained by a sorting process, initially managed by an edge input module 408 which receives messages, including edge information, from the instruction executor 300. It is possible to relax the sort to only regard the integral portion of each scan line intersection coordinate as significant. It is also possible to relax the sort further by only regarding each scan line intersection coordinate, clamped to the minimum and maximum X coordinates which are being produced by the current rendering process. Where appropriate, the edge input module 408 relays messages to modules 500, 600 and 700 downstream in the pipeline 22 via an output 498.

The edge input module 408 maintains references into and receives edge data from each of the three lists 402, 404, and 406. Each of these references is initialised to refer to the first edge in each list at the start of processing of a scan line. Thereafter, the edge input module 408 selects an edge record from one of the three referenced edge records such that the record selected is the one with the least X coordinate out of the three

referenced records. If two or more of the X-records are equal, each are processed in any order and the corresponding edge crossings output in the following fashion. The reference which was used to select that record is then advanced to the next record in that list. The edge just selected is formatted into a message and sent to an edge update module 410. Also, certain flags of the edge, in particular the current X, the priority numbers, and the direction flag, are formatted into a message which is forwarded to the activity determination and instruction generation module 500 as an output 498 of the edge processing module 400. Embodiments which use more or fewer lists than those described here are also possible.

Upon receipt of an edge, the edge update module 410 decrements the count of how many scan lines for which a current segment will last. If that count has reached zero, a new segment is read from the address indicated by the next segment address. A segment specifies:

(i) a value to add to the current X coordinate immediately the segment is read,

(ii) a new DX value for the edge,

(iii) a new DDX value for the edge, and

(iv) a new count of how many scan lines for which the new segment will last.

If there is no next segment available at the indicated address, no further processing is performed on that edge. Otherwise, the edge update module 410 calculates the X coordinate for the next scan line for the edge. This typically would involve taking the current X coordinate and adding to it the DX value. The DX may have the DDX value added to it, as appropriate for the type of edge being handled. The edge is then written into any available free slot in an edge pool 412, which is an array of two or more edge records. If there is no free slot, the edge update module 410 waits for a slot to become available. Once the edge record is written into the edge pool 412, the edge update module 410 signals via a line 416 to an edge output module 414 that a new edge has been added to the edge pool 412.

As an initial condition for the rendering of a scan line, the edge output module 414 has references to each of a next main edge list 420 and a next spill edge list 422, not seen in Fig. 4 but associated with the lists 404 and 406 in the edge record 32. Each of these references is initialised to the location where the, initially empty, lists 420 and 422 may be built up. Upon receipt of the signal 416 indicating that an edge has been added to the edge pool 412, the edge output module 414 determines whether or not the edge just added has a lesser X coordinate than the edge last written to the next main edge list 420 (if any).

If this is true, a "spill" is said to have occurred because the edge cannot be appended to the main edge list 404 without violating its ordering criteria. When a spill occurs, the edge is inserted into the next spill edge list 422, preferably in a manner that maintains a sorted next spill edge list 422. For example this may be achieve using a software sorting routine. In some embodiments spills may be triggered by other conditions, such as excessively large X coordinates.

If the edge added to the edge pool 412 has an X coordinate greater than or equal to the edge last written to the next main edge list 420 (if any), and there are no free slots available in the edge pool 412, the edge output module 414 selects the edge from the edge pool 412 which has the least X coordinate, and appends that edge to the next main edge list 420, extending it in the process. The slot in the edge pool 412 which was occupied by that edge is then marked as free.

Once the edge input module 408 has read and forwarded all edges from all three of its input lists 402, 404 and 406, it formats a message which indicates that the end of scan line has been reached and sends the message to both the activity determination and instruction generation module 500 and the edge update module 410. Upon receipt of that message, the edge update module 410 waits for any processing it is currently performing to complete, then forwards the message to the edge output module 414. Upon receipt of the message, the edge output module 414 writes all remaining edge records from the edge pool 412 to the next main edge list 404 in X order. Then, the reference to the next main edge list 420 and the main edge list 404 are exchanged between the edge input module 408 and the edge output module 414, and a similar exchange is performed for the next spill edge list 422 and the spill edge list 406. In this way the initial conditions for the following scan line are established.

Rather than sorting the next spill edge list 422 upon insertion of edge records thereto, such edge records may be merely appended to the list 422, and the list 422 sorted at the end of the scan line and before the exchange to the current spill list 406 becomes active in edge rasterisation of the next scan line. Other methods of sorting the edges involving fewer or more lists may be used, as well as different sorting algorithms.

It can be deduced from the above that edge crossing messages are sent to the activity determination and instruction generation module 500 in scan line and pixel order (that is, they are ordered firstly on Y and then on X) and that each edge crossing message is labelled with the priority to which it applies.

Fig. 12A depicts a specific structure of an active edge record 418 that may be created by the edge processing module 400 when a segment of an edge is received. If the first segment of the edge is a step (orthogonal) segment, the X-value of the edge is added to a variable called "X-step" for the first segment to obtain the X position of the activated edge. Otherwise, the X-value of the edge is used. This means that the edges in the new edge record must be sorted by $X_{edge} + X_{step}$. The X_{step} of the first segment should, therefore, be zero, in order to simplify sorting the edges. The Y-value of the first segment is loaded into the NY field of the active edge record 418. The DX field of the active edges copied from the DX field identifier of vector or quadratic segments, and is set to zero for a step segment. A u-flag as seen in Fig. 12A is set if the segment is upwards heading (see the description relating to Fig. 13A). A q-flag is set if the segment is a quadratic segment, and cleared otherwise. An i-flag is provided and is set if the segment is invisible. A d-flag is set when the edge is used as a direct clipping object, without an associated clipping level, and is applicable to closed curves. The actual priority level of the segment, or a level address is copied from the corresponding field of the new edge record into a level (ADDR) field in the active edge record 418. A segment address/DDX field of the active edge record 418 is either the address of the next segment in the segment list or copied from the segment's DDX value, if the segment is quadratic. The segment address is used to terminate an edge record. As a consequence, in the preferred embodiment, any quadratic curve (ie: that uses the DDX filed) will be a terminal segment of an edge record.

It will be appreciated from Fig. 12A that other data structures are also possible, and necessary for example where higher-order polynomial implementations are used. Further, the segment address and the DDX field may be separated into different fields, and additional flags provided to meet alternate implementations.

Fig. 12B depicts the arrangement of the edge records described above in the preferred embodiment and used in the edge processing module 400. The edge pool 412 is supplemented by a new active edge record 428, a current active edge record 430 and a spill active edge record 432. As seen in Fig. 12B, the records 402, 404, 406, 420 and 422 are dynamically variable in size depending upon the number of edges being rendered at any one time. Each record includes a limit value which, for the case of the new edge list 402, is determined by a SIZE value incorporated with the LOAD_EDGES_AND_RENDER instruction. When such an instruction is encountered,

SIZE is checked and if non-zero, the address of the new edge record is loaded and a limit value is calculated which determines a limiting size for the list 402.

Although the preferred embodiments utilises arrays and associated pointers for the handling of edge records, other implementations, such as linked lists for example may be used. These other implementations may be hardware or software-based, or combinations thereof.

The specific rendering of the image 78 shown in Fig. 8 will now be described with reference to scan lines 34, 35 and 36 shown in Fig. 10. In this example, the calculation of the new X co-ordinate for the next scan line is omitted for the purposes of clarity, with Figs. 12C to 12I illustrating the output edge crossing being derived from one of the registers 428, 430 and 432 of the edge pool 412

Fig. 12C illustrates the state of the lists noted above at the end of rendering scan line 34 (the top portion of the semi-transparent blue triangle 80). Note that in scan line 34 there are no new edges and hence the list 402 is empty. Each of the main edge lists 404 and next main edge list 420 include only the edges 82 and 84. Each of the lists includes a corresponding pointer 434, 436, and 440 which, on completion of scan line 34, point to the next vacant record in the corresponding list. Each list also includes a limit pointer 450, denoted by an asterisk (*) which is required to point to the end of the corresponding list. If linked lists were used, such would not be required as linked lists include null pointer terminators that perform a corresponding function.

As noted above, at the commencement of each scan line, the next main edge list 420 and the main edge list 404 are swapped and new edges are received into the new edge list 402. The remaining lists are cleared and each of the pointers set to the first member of each list. For the commencement of scan line 35, the arrangement then appears as seen in Fig. 12D. As is apparent from Fig. 12D, the records include four active edges which, from Fig. 10, are seen to correspond to the edges 92, 94, 84 and 82.

Referring now to Fig. 12E, when rendering starts, the first segment of the new edge record 402 is loaded into an active edge record 428 and the first active edge records of the main edge list 404 and spill edge list 406 are copied to records 430 and 432 respectively. In this example, the spill edge list 406 is empty and hence no loading takes place. The X-positions of the edges within the records 428, 430 and 432 are then compared and an edge crossing is emitted for the edge with the smallest X-position. In this case, the emitted edge is that corresponding to the edge 92 which is output together with its priority value. The pointers 434, 436 and 438 are then updated to point to the next record in the list.

The edge for which the edge crossing was emitted is then updated (in this case by adding DX = 0 to its position), and buffered to the edge pool 412 which, in this example, is sized to retain three edge records. The next entry in the list from which the emitted edge arose (in this case list 402) is loaded into the corresponding record (in this case record 428). This is seen in Fig. 12F.

Further, as is apparent from Fig. 12F, a comparison between the registers 428, 430 and 432 again selects the edge with the least X-value which is output as the appropriate next edge crossing (X=85, P=2). Again, the selected output edge is updated and added to the edge pool 412 and all the appropriate pointers incremented. In this case, the updated value is given by $X \leftarrow X + DX$, which is evaluated as $84 = 85 - 1$. Also, as seen, the new edge pointer 434 is moved, in this case, to the end of the new edge list 402.

In Fig. 12G, the next edge identified with the lowest current X-value is again that obtained from the register 430 which is output as an edge crossing (X=115, P=2). Updating of the edge again occurs with the value be added to the edge pool 412 as shown. At this time, it is seen that the edge pool 412 is now full and from which the edge with the smallest X-value is selected and emitted to the output list 420, and the corresponding limited pointer moved accordingly.

As seen in Fig. 12H, the next lowest edge crossing is that from the register 428 which is output (X=160 P=1). The edge pool 412 is again updated and the next small X-value emitted to the output list 420.

At the end of scan line 35, and as seen in Fig. 12I, the contents of the edge pool 412 are flushed to the output list 420 in order of smallest X-value. As seen in Fig. 12J, the next main edge list 420 and the main edge list 404 are swapped by exchanging their pointers in anticipation of rendering the next scan line 36. After the swapping, it is seen from Fig. 12J that the contents of the main edge list 404 include all edge current on scan line 36 arranged in order of X-position thereby permitting their convenient access which facilitates fast rendering.

Ordinarily, new edges are received by the edge processing module 400 in order of increasing X-position. When a new edge arrives, its position is updated (calculated for the next scan line to be rendered) and this determines further action as follows:

(a) if the updated position is less than the last X-position output on the line 498, the new edge is insertion sorted into the main spill list 406 and the corresponding limit register updated;

(b) otherwise, if there is space, it is retained in the edge pool 412.

As is apparent from the forgoing, the edge pool 412 aids in the updating of the lists in an ordered manner in anticipation of rendering the next scan line in the rasterised image. Further, the size of the edge pool 412 may be varied to accommodate larger numbers of non-ordered edges. However, it will be appreciated that in practice the edge pool 412 will have a practical limit, generally dependent upon processing speed and available memory with the graphic processing system. In a limiting sense, the edge pool 412 may be omitted which would ordinarily require the updated edges to be insertion sorted into the next output edge list 420. However, in the preferred embodiment this situation is avoided, as a normal occurrence through the use of the spill lists mentioned above. The provision of the spill lists allows the preferred embodiment to be implemented with an edge pool of practical size and yet handle relatively complex edge intersections without having to resort to software intensive sorting procedures. In those small number of cases where the edge pool and spill list are together insufficient to accommodate the edge intersection complexity, sorting methods may be used.

An example of where the spill list procedure is utilised is seen in Fig. 13A where three arbitrary edges 60, 61 and 63 intersect an arbitrary edge 62 at a relative position between scan lines A and B. Further, the actual displayed pixel locations 64 for each of scan lines A, B, are shown which span pixel locations C to J. In the above described example where the edge pool 412 is size to retain three edge records, it will be apparent that such an arrangement alone will not be sufficient to accommodate three edge intersections occurring between adjacent scan lines as illustrated in Fig. 13A.

Fig. 13B shows the state of the edge records after rendering the edges 60, 61 and 63 on scan line. The edge crossing H is that most recently emitted and the edge pool 412 is full with the updated X-values E, G and I for the edges 60, 61 and 63 respectively for the next scan line, scan line B. The edge 62 is loaded into the current active edge record 430 and because the edge pool 412 is full, the lowest X-value, corresponding to the edge 60 is output to the output edge list 420.

In Fig. 13C, the next edge crossing is emitted (X = J for edge 62) and the corresponding updated value determined, in this case X = C for scan line B. Because the new updated value X = C is less than the most recent value X = E copied to the output list 420, the current edge record and its corresponding new updated value is transferred directly to the output spill list 422.

Fig. 13D shows the state of the edge records at the start of scan line B where it is seen that the main and output lists, and their corresponding spill components have been

swapped. To determine the first emitted edge, the edge 60 is loaded into the current active edge register 430 and the edge 62 is loaded into the spill active edge register 432. The X-values are compared and the edge 62 with the least X-value (X = C) is emitted, updated and loaded to the edge pool 412.

Edge emission and updating continues for the remaining edges in the main edge list 404 and at the end of the scan line, the edge pool 412 is flushed to reveal the situation shown in Fig. 13E, where it is seen that each of the edges 60 to 63 are appropriately ordered for rendering on the next scan line, having been correctly emitted and rendered on scan line B.

As will be apparent form the forgoing, the spill lists provide for maintaining edge rasterisation order in the presence of complex edge crossing situations. Further, by virtue of the lists being dynamically variable in size, large changes in edge intersection numbers and complexity may be handled without the need to resort to sorting procedures in all but exceptionally complex edge intersections.

In the preferred embodiment the edge pool 412 is sized to retain eight edge records and, size of the lists 404, 420 together with their associated spill lists 406, 422, is dynamically variable thereby providing sufficient scope for handling large images with complex edge crossing requirements.

3.0 Activity Determination and Instruction Generation Module

Expression trees are often used to describe the compositing operations required to form an image, and typically comprise a plurality of nodes including leaf nodes, unary nodes and binary nodes. A leaf node is the outermost node of an expression tree, has no descendent nodes and represents a primitive constituent of an image, namely a graphic object. Unary nodes represent an operation, which modifies the pixel data of the object coming out of the part of the tree below the unary operator. A binary node typically branches to left and right sub-trees, wherein the sub-trees each comprise at least one leaf node. The binary node represents an operation between one object on one branch and another object on another branch.

Examples of such operations are described in "Compositing Digital Images", Porter, T; Duff, T; Computer Graphics, Vol. 18 No. 3 (1984) pp. 253-259. Some of these Porter and Duff compositing operations are shown in Fig 15. For ease of illustration, the graphic objects shown in Fig. 15 are fully opaque. As can be seen, a number of these operators (e.g. out) do not provide data (e.g. colour data) in the centre overlapping region of the two objects. Generally, these operators (e.g. out) will always provide data

(e.g. colour data) in the centre overlapping region of the two objects if one or both of the two objects are partially transparent. However, there are special cases where particular operators provide no data in the centre overlapping region of the two objects if one or both of the two objects are fully opaque.

Turning now to Fig 19, there is shown a typical example of a simple expression tree and a corresponding instruction list. The expression tree comprises five leaf nodes 10, 9, 7, 6 and 0-5 describing the graphic objects A, B, C, D and PAGE respectively. The expression tree also contains nodes 8, 11, 12, and 13 each having two branches and representing the operations "out", "in", "over" and "over" respectively. The instruction list contains a list of instructions for rendering the expression tree. However, problems arise when generating an instruction list for a given pixel location because the compositing operation depends on which of the graphic objects composited by the operation are active at that given pixel location. The preferred Activity Determination and Instruction Generation Module 500 seeks to solve these problems.

The operation of the Activity Determination and Instruction Generation Module 500 will now be described with reference to Fig. 5. The Instruction Generator 300 during an initialisation phase passes to the Level Activation Table Generator 502 an object graphic description of the image to be rendered. This object graphic description is in the form of an expression tree, such as for example the expression tree shown in Fig. 19. The Level Activation Table Generator 502 then generates a Level Activation Table, which is a linear form of the expression tree and stores it the Level Activation Table Store 34. The generated Level Activation Table contains a plurality of records, one record for each binary operator node, unary node and leaf node of the expression tree.

In addition, the Level Activation Table also contains fields for storing data on certain inherent properties of the binary operators, and fields for storing data on the activity of the branches of the binary operator nodes. The data concerning the properties of the binary operators is static, that is dependent on the actual operators used and not on the location of the currently scanned pixel. Consequently, this data can be generated and stored in the level activation table during the initialisation phase. However, the data concerning branch activity is dependent upon the currently scanned pixel location. During the initialisation phase, as there is no currently active pixel location, the branch activity data of the Level Activation Table is initialised as inactive.

The Pixel Sequential Rendering Apparatus 20 then commences scanning the pixel locations of the image in a raster scan order. The edge processing module 400 passes to

the Level Activation Table Update Module 504, the pixel locations of the edges of the current scan line being scanned. The Level Activation Table Update Module 504 accesses the Level Activation Table from the store 34 and updates the Table depending upon which pixel location is currently being scanned. For ease of implementation, the Table is updated only once for those pixel locations between adjacent edges. This is based on the recognition that the instruction list for rendering the expression tree is the same for the group of pixel locations between any two adjacent edges of a scan line. Alternatively, the Table can be updated for each scanned pixel location.

The Traversal and Instruction Generator Module 506, then generates instructions based on this updated Level Activation Table which instructions are then passed onto the Fill Colour Determination Module 600. This process is continued for each group of pixel locations between adjacent edges until the image is rendered.

Before proceeding with any further with the description of the preferred embodiments, a brief review of terminology used herein is discussed.

Turning now to Figs. 18A and 18B, there is shown a binary operation (illustrated as an expression tree) between a source object L and a destination object R. The expression tree has a binary node OP with a left branch to the object L and right branch to the object R. An object is considered to be active at a pixel if the currently scanned pixel is inside the boundary edges, which apply to the object. If the pixel is outside the boundary edges then the object is inactive at that pixel. Thus the object will be active during the scanning of those group of pixels inside the boundary edges. The branch of an operator to an object is considered to be active if the object is active.

Regardless of the actual operation being performed, the binary operation of Fig. 18A resolves into four regions of activity as indicated below:

Region 1. L object active, R object inactive ($L \cap \overline{R}$);

Region 2. L object active, R object active ($L \cap R$);

Region 3. L object inactive, R object active ($\overline{L} \cap R$); and

Region 4. L object inactive, R object inactive ($\overline{L \cup R}$).

Region 4 always results in no operation (*NOP*) being required to be performed and as a consequence, there exists three different combinations of active levels for a binary tree. For ease of explanation, the region ($L \cap \overline{R}$) is referred to herein as the left node region, or left region, the region ($\overline{L} \cap R$) is referred to herein as the right node region, or right region, and the region ($L \cap R$) is referred to herein as the common region. The left

node region and the right node region are associated with the left and right branches of the parent node respectively and correspond to the objects operator order (eg L op R) and not their location in the image.

For the purposes of this description, a binary operator is considered to be *active* if it provides data up the tree. The activity of a binary operator is an inherent property of the operator itself and the activity of its left and right branches.

As an example, compare the "out" operator of Fig. 15 with Fig. 18A. The "out" operator has the inherent property of passing data up the tree for the region $(L \cap \overline{R})$ irrespective of whether the top (left branch) object is partially transparent or opaque; and in the common region $(L \cap R)$ only if the lower (right branch) object is partially transparent. In the former case, this will only occur when its left branch becomes active and right branch inactive for the currently scanned group of pixels. In the latter case, this will only occur when the right and left branches are both active for the currently scanned group of pixels. On the other hand, the "out" operator has the inherent property of passing no data up the tree for the region $(\overline{L} \cap R)$, irrespective of whether the lower (right branch) object is partially transparent or opaque.

As another example, compare the "in" operator of Fig. 15 with Fig. 18A. The "in" operator has the inherent property of passing data (eg colour data) up the tree for the region $(L \cap R)$. However, this will only occur when its left and right branches are active. On the other hand, the "in" operator has the inherent property of passing no data up the tree for the regions $(\overline{L} \cap R)$ and $(L \cap \overline{R})$.

As another example, compare the "rout" operator of Fig. 15 with Fig. 18A. The "rout" operator has the inherent property of passing data up the tree for the region $(\overline{L} \cap R)$, irrespective of whether the top (right branch) object is partially transparent or opaque; and in the common region $(L \cap R)$ only if the lower (left branch) object is partially transparent. In the former case, this will only occur when its left branch becomes active and right branch inactive for the currently scanned group of pixels. In the latter case, this will only occur when the right and left branches are both active for the currently scanned group of pixels. On the other hand, the "rout" operator has the inherent property of passing no data up the tree for the region $(L \cap \overline{R})$, irrespective of whether the lower (left branch) object is partially transparent or opaque.

It should be noted that there is a distinction between the performance of a binary operator and its activity. A binary operator will only *perform* an operation when both its branches are active. Otherwise, the result is taken directly from the active branch, if required by the operator.

The Activity Determination and Instruction Generation Module 500 traverses and generates instructions depending upon of the activity of the binary operators, the activity of the branches of the expression tree, and the activity of the graphic objects (leaf nodes).

The manner in which these instructions are generated will be described with reference to the example shown in Fig. 20. Fig. 20 shows the expression tree of Fig. 19, where object C is active and shows a corresponding instruction list generated by the Activity Determination and Instruction Generation Module 500. The Module 500 initially determines the aforementioned inherent properties of the operators of the expression tree. These properties are permanently stored (not shown) for retrieval and use by the module 500. In the example shown in Fig. 20, the Module 500 retrieves the binary operator properties of the "over", "in", and "out" operators from storage. These properties are stored in logic form. For example, the properties of the "over" operator are stored as $(\bar{L}\cap R)$=TRUE and $(L\cap\bar{R})$=TRUE. Namely, the "over" operator has the capability of passing data up the tree for these regions. In another example, the properties of the "in" operator are stored as $(\bar{L}\cap R)$=FALSE and $(L\cap\bar{R})$=FALSE. Namely, the "in" operator does not have the capability for passing data up the tree for these regions. In still another example, the properties of the "out" operator are stored as $(\bar{L}\cap R)$=TRUE and $(L\cap\bar{R})$=FALSE. As will become apparent the inherent property of the binary operator in the region $(L\cap R)$ is not necessary for the performance of the Module 500 and is not stored in memory, as the binary operation of the node will always be performed when the right and left branches are active.

In the example of Fig. 20, the Activity Determination and Instruction Generation Module 500 after the initialisation phase then determines for a particular group of scanned pixel locations, that object C is active and Objects A, B, and D are inactive. The Module 500 determines the activity of the branches of each of the binary nodes from the activity of the objects. Namely, all the branches directly coupling the active leaf node objects (e.g. Object C and PAGE) to the root node (eg. 13) will be active. In this description, an active left branch is designated as L Active = TRUE, an inactive left branch is designated as L Active = FALSE, an active right branch is designated as R Active = TRUE, and an

inactive right branch is designated as R Active = FALSE. In the example of Fig. 20, the Module 500 determines that:

(i) the left and right branches of binary node 13 are active (ie. L Active = TRUE and R Active = TRUE);

(ii) the left branch is inactive and the right branch is active of binary node 12 (ie L Active = FALSE and R Active = TRUE);

(iii) the left and right branches of binary node 11 are inactive (ie L Active = FALSE and R Active = FALSE); and

(iv) the left branch is active and the right branch is inactive of binary node 8 (ie. L Active = TRUE and R Active = FALSE).

For the purposes of this description, the symbols && , || , and ! as used herein refer to the binary logic operators AND, OR and NOT respectively.

The Activity Determination and Instruction Generation Module 500 then traverses and generates instructions for the current group of scanned pixel locations depending upon of the activity of the binary operators, the activity of the branches of the expression tree, and the activity of the graphic objects (leaf nodes). The Module 500 traverses the expression tree commencing at the root node (eg. node 13) in a top down - left to right manner.

The Module 500 traverses only those branches of the expression tree that satisfy the following conditions:

(1) the left branch of any previously traversed binary node is traversed to its descendent node if (L active && R active) || ($(L \cap \bar{R})$ && L active && !R active) ==TRUE for the previously traversed binary node.

(2) the right branch of any previously traversed binary node is traversed to its descendent node if (L active && R active) || ($(\bar{L} \cap R)$ && !L active && R active)==TRUE for the previously traversed binary node.

The Activity Determination and Instruction Generation Module 500 generates during the same time as the traversal, operator instructions for any of the traversed binary nodes having active right and left branches, and leaf value instructions for any of the traversed leaf node.

In the example of Fig. 20, the Activity Determination and Instruction Generation Module 500 begins its traversal at root node 13. As both the left and right branches of the root node 13 are active, the Module 500 generates an "over" operator instruction. For the same reasons, the Module 500 will traverse to binary node 12 and binary node 0-5. As

the Module 500 traverses in a top down left to right manner, it first traverses to binary node 12. At this binary node 12, the Module 500 does not generate an "over" operation, as the left branch of the binary 12 is inactive. For the same reasons, the Module 500 does not traverse the left branch to binary node 11. However, as the right branch of binary node 12 is active and the left branch of binary node 12 is inactive and as $(\bar{L} \cap R) = TRUE$ for the "over" operator, the Module 500 traverses to binary node 8. At this binary node 8, the Module 500 does not generate an "out" operation, as the right branch of the binary 8 is inactive. For the same reasons, the Module 500 does not traverse the right branch to leaf node 6. However, as the left branch of binary node 8 is active and the right branch of binary node 8 is inactive and as $(L \cap \bar{R}) = TRUE$ for the "out" operator, the Module 500 traverses to leaf node 7, where the Module 500 generates a leaf value instruction C. The Module 500 then returns to root node 13 to traverse to leaf node 0-5, where the Module 500 generates a leaf value instruction PAGE. Comparing Figures 19 and 20, it can thus be seen that the Module 500 generates a minimal instruction set corresponding to the expression tree for that current group of scanned pixel locations. The Module 500 will then repeat the operation for the next group of scanned pixel locations between the next adjacent edges.

The operation of the components of the Activity Determination and Instruction Generation Module 500 will now be described in more detail with reference to Fig. 5.

As mentioned above, the Level Activation Table Generator 502 generates a Level Activation Table. Turning now to Fig. 21, there is shown an example of such a generated Level Activation Table. This particular Level Activation Table represents a linearised form of the expression tree shown in Fig. 19. The Level Activation Table of Fig. 21 has a record for each node of the expression tree. These records each have the following fields "Index", "L Active", "R Active", "$(L \cap \bar{R})$", "$(\bar{L} \cap R)$", "Leaf/Operator Entry", "Node Active", "Parent", "Node is L", "Generate L", "Generate R", "$(L \cap R)$ op used", and "R Branch Index". The contents of the fields "Index", "$(L \cap \bar{R})$", "$(\bar{L} \cap R)$", "Leaf/Operator Entry", "Parent", "Node is L", and "R Branch Index" are static in that they do not vary as a function of the current scanned pixel location. On the other hand, the contents of the fields "L Active", "R Active", "Node Active", "Generate L", "Generate R", and "$(L \cap R)$ op used" may vary depending upon the currently scanned pixel location. The latter fields are updated by the Level Activation Table Update Module 504, for each group of pixel locations between the adjacent edges.

The "Index" field contains a numeric label of the node associated with the relevant record. For example, the record corresponding to the node 12 of Fig. 19 has an "Index" field set to 12. The "Parent" field contains the numeric label of the parent node of the node associated with the record. For example, the record corresponding to the node 12 of Fig. 19 has the "Parent" field set to 12. Namely, the parent node of node 12 is node 13. The "R branch Index" field contains a numeric label of the node descendent on the right branch of the node associated with the record. For example, the record corresponding to the node 12 of Fig. 19 has the "R branch Index" field set to 8. Namely, the node on the right branch of node 12 is node 8. The "Node is L" field is a logic field indicating whether or not the node associated with the record is descendant from its parent node's left branch. For example, the record corresponding to node 12 has the "Node is L" field set to TRUE (1). Namely, node 12 is descendant from parent node 13 via the parent node's 13 left branch. The "Leaf/Operator Entry" field contains the binary operator of the binary node or the object of the leaf node, which is associated with the relevant record. For example, the record corresponding to the binary node 12 of Fig. 19 has the "Leaf/Operator Entry" field set to the compositing operator "over". In another example, the record corresponding to the leaf node 7 of Fig. 19 has the "Leaf/Operator Entry" field set to object "C". In this way, the structure of the expression tree can be fully reconstructed.

The "L Active" field is a logic field indicating whether the left branch of the binary node corresponding to the record is active or not depending upon the current group of scanned pixels. Similarly, the "R Active" field is a logic field indicating whether the right branch of the binary node corresponding to the record is active or not depending upon the current group of scanned pixels. During the initialisation phase there are no currently scanned pixels, thus all "L Active" and "R Active" fields of the binary nodes can be set to FALSE (0), with the possible exception of the "R Active" field of the root node. For example, the "R Active" field of the record corresponding to the root node 13 of Fig. 19 is set to TRUE (1), as the right branch of the node 13 is always active irrespective of the scanned pixel. It should be noted that as leaf nodes have no branches, there is no need to set the "L Active" and "R Active" fields for the corresponding records.

The "$(L \cap \overline{R})$" field is a logic field indicating whether or not data from this region $(L \cap \overline{R})$ is required to be passed up the expression tree. Similarly, the "$(\overline{L} \cap R)$" field is a logic field indicating whether or not data from this region $(\overline{L} \cap R)$ is required to be passed

up the expression tree. As mentioned above, this is an inherent property of the compositing operator. For example, the record corresponding to node 12 of Fig. 19 has the field "Leaf/Operator Entry" set to "over" and has the associated fields "$(L \cap \bar{R})$"and "$(\bar{L} \cap R)$" both set to TRUE (1), as data from both these regions are required to be passed up the tree. In another example, the record corresponding to node 11 of Fig. 19 has "Leaf/Operator Entry" set to "in" and has the associated fields "$(L \cap \bar{R})$"and "$(\bar{L} \cap R)$" both set to FALSE (0) as no data from these regions are required to be passed up the tree. It should be noted that as these fields relate to binary nodes only (ie compositing operations) there is no need to set these fields for records corresponding to leaf nodes.

The "Node Active" field is a logic field which is set to TRUE (1) when the following condition is satisfied otherwise it is set to FALSE (0):

$(L \cap \bar{R}$ && L active && !R active)
||
(L active && Ractive)
||
$(\bar{L} \cap R$ && !L active && R active)
== TRUE

For example, the fields "Node Active" for binary nodes 8,11, and 12 shown in Fig. 19 are set to FALSE (0) during the initialisation phase as all the corresponding branches (L Active and R Active) are inactive (ie. FALSE). During the initialisation phase, the "Node Active" field for binary node 13 of Fig. 19 is set to TRUE (1) as its right branch is active (R Active = TRUE) and its left branch inactive (L Active = FALSE) and the region $\bar{L} \cap R$ is required for the operation of the operator "over". It should be noted that as this field relates to binary nodes only (ie compositing operations) there is no need to set these fields for records corresponding to leaf nodes. The "Node Active" field of a record indicates whether or not the corresponding binary operator is active.

The "Generate L" field is a logic field which is set to TRUE (1) when the following condition is satisfied otherwise it is set to FALSE (0):

$(L \cap \bar{R}$ && L active && !R active)
||
(L active && Ractive)
== TRUE

For example, the fields "Generate L" for all the binary nodes 13, 12, 11, and 8 of Fig. 19 are set during the initialisation phase to FALSE (0) as this condition is not met by any of these nodes. Namely, the left branches of nodes 8, 11, 12, and 13 are all inactive ie L Active = FALSE (0). It should also be noted that as this field relates to binary nodes only (ie compositing operations) there is no need to set these fields for records corresponding to leaf nodes.

Similarly, the "Generate R" field is a logic field which is set to TRUE (1) when the following condition is satisfied otherwise it is set to FALSE (0):

(L active && Ractive)

||

($\overline{L} \cap R$ && !L active && R active)

== TRUE

For example, the fields "Generate R" for the binary nodes 12, 11, and 8 of Fig. 19 are set during the initialisation phase to FALSE (0) as this condition is not met by any of these nodes. For instance, the right branches of nodes 8, 11, and 12 are all inactive ie R Active = FALSE (0). However, the field "Generate R" for the binary node 13 is set during the initialisation to TRUE (1) as the right branch is active, the left branch inactive and ($\overline{L} \cap R$) = TRUE. It should also be noted that as this field relates to binary nodes only (ie compositing operations) there is no need to set these fields for records corresponding to leaf nodes.

The field "($L \cap R$) op used" is a logic field which is set to TRUE (1) when the following condition is satisfied otherwise it is set to FALSE (0):

(L active && Ractive) ==TRUE

For example, as all the left branches of the binary nodes of Fig. 19 are inactive during the initialisation phase this field is set to FALSE (0) for all the binary nodes.

Turning now to Fig. 22, there is shown a logic circuit for setting the aforementioned fields "Node Active", "Generate L", "Generate R", and "($L \cap R$) op used" of the Level Activation Table. This logic circuit contains a series of logic gates shown in conventional format, which is self-explanatory and need not be explained further.

Returning now to Fig. 5, the Level Activation Table Generator Module 502 stores the Level Activation Table generated during the initialisation phase in the memory 34. The Update Module 504 then retrieves this initialised Level Activation Table for each group of scanned pixel locations between adjacent edges and updates the fields of the

Table. The Update Module 504 changes the state of the fields "L Active", "R Active", "Node Active", "Generate L", "Generate R", and "(L∩R) op used" depending upon the current group of scanned pixel locations. The Update Module 504 updates the records in a predetermined manner commencing at those records of the parent nodes of the leaf nodes corresponding to the active objects for that pixel location. Whenever the Update Module 504 changes the state of the "Node Active" field for a record, the Update Module 504 triggers a corresponding change in the state of the "L active" or "R active" field in the record of the parent node, depending on the state of the "Node is L" field. The Update Module 504 then updates the remaining fields of the record of the parent node in accordance with this newly changed "L Active" or "R Active field". This Updating process continues until a level is reached where the "Node Active" field remains unchanged. In the event there are more than one active objects, the Update Module updates the table one active object at a time.

The updating process of the Update Module 504 will now be explained with reference to Fig. 23, which shows an updated Level Activation Table for the expression tree of Fig. 20 where object C is active. The Update Module 504 first determines that the Object C is active for the current group of scanned pixel locations and then retrieves the initialised Level Activation Table from memory 34. The Update Module 504 determines the parent node of the active object C from the "Parent" field of the record of leaf node C. The update Module then proceeds to update the records of the table in the following manner commencing with the parent node of object C (record 8):

- the "L Active" field in the parent node (record 8) is set;
- this asserts the "Generate L" field in record 8;
- the "Node Active" field is set in record 8;
- the "R active" field in the parent node (record 12) is set;
- this asserts the "Generate R" field in record 12;
- the "Node Active" field is set in record 12;
- the "L Active" field is set in the parent node (record 13);
- this asserts the "Generate L" and "L∩R op used" fields in record 13; and
- the "Node Active" field in record 13 is already set, so processing stops.

At this point, the table is in a correct state for generating instructions.

Returning now to Fig. 5, the Update Module 504 stores this updated Level Activation Table for the current group of scanned pixel locations in memory 34 which is then retrieved by the Traversal and Instruction Generator Module 506. The Traversal and

Instruction generator Module 506 is a stack-machine based robot, which traverses the expression tree top-down, generating instructions in accordance with the updated Level Activation Table. The stack is used to store the "R branch Index" of a record. The stack-machine performs the following operations in accordance with the following pseudo-code:

```
IF Node is operation
THEN
  IF L or R op used flag is set
  THEN
    add operation to instruction list
  ENDIF
  IF Generate R flag is set
    Push R branch index on the stack
  ENDIF
  IF Generate L flag is set
    Process next entry
  ENDIF
ELSE (Node is a leaf)
  Add leaf value instruction to instruction list
ENDIF
IF stack is empty
THEN
  STOP
ELSE
  Pop index off stack
  Process the entry
ENDIF
```

The traversal and instruction generating process of the Instruction Module 506 will now be explained with reference to Fig. 23, which shows an updated level activation table for the expression tree of Fig. 20 where object C is active. The generation of instructions in accordance with the above pseudo-code and the Level Activation Table of Fig. 23 proceeds as follows:

(i) processing starts at record 13 (root node);

(ii) the field "L∩R op used" of record 13 is TRUE (1) so the "over" instruction is added to the instruction list;

(iii) the field "Generate R" of record 13 is TRUE (1), so the contents of the "R Branch Index" of record 13 is added to the stack. The stack thus storing the index (5);

(iv) the field "Generate L" of record 13 is TRUE (1), so processing moves to record 12;

(v) the field "L∩R op used" of record 12 is FALSE (0), so no instruction is added;

(vi) the field "Generate R" of record 12 is TRUE (1), so the contents of the "R Branch Index" of record 12 is added to the stack. The stack currently thus storing the indices (8,5);

(vii) the field "Generate L" of record 12 is FALSE (0), thus the processing falls through;

(viii) the stack is not empty, so index 8 is popped off the stack, and processing moves to the record corresponding to index 8. The stack thus currently storing the index (5);

(ix) the field "L∩R op used" of record 8 is FALSE (0), so no instruction is added;

(x) the field "Generate R" of record 8 is FALSE (0), so nothing is added to the stack. The stack thus currently storing the index (5);

(xi) the field "Generate L" of record 8 is TRUE (1), so processing moves to the next entry, i.e record 7;

(xii) the record 7 is a leaf, so a leaf value instruction is added to the list, and processing falls through;

(xiii) stack is not empty, so index 5 is popped off the stack, and processing moves to index 5. The stack is now empty.

This process proceeds on to completion.

The instructions generated for this example are shown in Fig. 20.

In this way, the Traversal and Instruction Generator Module 506 is able to generate a minimal set of instructions necessary for the rendering of the expression tree for any one or more active objects. These instructions are then passed onto the fill colour determination module 600.

In another embodiment of the Activity Determination and Instruction Generation Module 500, the module takes into account the fact that, in the overlapping region, an

opaque object may obscure the other object. Indeed, either branch of a node can be obscured by the other. If an object is obscured by an opaque object, then it does not need to be generated when both objects are active. The Level Activation Table can be modified to prevent these redundant operations from being performed.

Turning now to Fig. 25, there is shown an example of such a modified Level Activation Table. This Table is the same structure as the Level Activation Table of Fig. 21, except it has two extra fields "L obscures R" and "R obscures L". The field "L obscures R" is a logic field and indicates whether or not the left object obscures the right object when both the objects are active. Similarly, the field "R obscures L" is a logic field and indicates whether or not the right object obscures the left object when both the objects are active. The data contained in the fields "Node Active", "Generate R", "L∩R op used", and "Generate L" is also based on different logic than the Level Activation Table of Fig. 21. The obscurance flags are very useful in expression trees comprising both opaque and transparent objects.

Turning now to Fig. 24, there is shown a logic circuit for setting the fields "Node Active", "Generate L", "Generate R", and "(L∩R) op used" of the modified Level Activation Table. It can seen that the "Node Active" logic has changed from Fig. 22, and is now based on whether data will be generated by the node. The critical path of the circuit is longer, but this allows clipping of objects to be performed using the same logic. If both of the obscurance fields are set, no data is generated in the intersection region (this is part of the implementation of a CLIP OUT).

These obscurance fields may be used to implement a CLIP IN operation and a CLIP OUT operation, as shown in the Level Activation Table of Fig. 25. The field settings for the CLIP IN and CLIP OUT operation are shown in the Table of Fig. 25.

The difference between a CLIP IN operation and a simple in operation is that the right branch operation is never performed. This can be seen from a comparison of the truth table of the IN operation as shown in Fig. 26 and from the truth table of the CLIP IN operation as shown in Fig. 27. In these truth tables, T stands for transparent (inactive), O stands for opaque, and L and R stand for L opacity unknown and R opacity unknown. The cases where a difference occurs are highlighted. In the CLIP IN case, activation of the right object enables the generation of instructions for the left object, such that the left object is edge clipped. Note that, the right object may be a composition of arbitrarily many objects, using whatever combinations and different fill rules as may be required. The activity state of right object is all that is used by the CLIP IN operation. In previous

systems a clipping object consist of a separate single level which when activated or de-activated effected a counter for all levels to which the clip applies. In the present arrangement, the clip object appears as a special case of a general drawing object and appears as a node of the expression tree and consequently the activating or de-activating applies the clip without the need to increment or decrement counters on every level.

The difference between a CLIP OUT operation and a simple out operation can be seen from a comparison of the truth table of the OUT operation as shown in Fig. 28 and from the truth table of the CLIP IN operation as shown in Fig. 29. Again, T stands for transparent (inactive), O stands for opaque, and L and R stand for L opacity unknown and R opacity unknown. The cases where a difference occurs are again highlighted. In the CLIP OUT case, activation of the right object prevents the generation of instructions for the left object, such that the left object is edge clipped. As for a CLIP IN, the right object may be an arbitrarily complex collection of objects, with different fill rules as required. It is only the activity state that is used for clipping.

The Activity Determination and Instruction Generation Module 500 describe above is based on an expression tree. However, the principles of the Module 500 may be generalised to DAGs (Directed Adjacency Graphs). This can be achieved by a further embodiment of the Module 500 by allowing the parent node field of a record of the Level Activation Table to contain a list of table entries to its parent nodes, and providing an L index pointer. Changing the state of a node would then require that all of its parent nodes be modified, and instruction generation would be required to follow the L index, rather than simply looking for the next table entry. DAGs would be useful for use with clipping objects, where multiple objects are clipped by the same object.

A still further embodiment is based on the fact that the starting node, i.e. the first node which will provide an operation or data, will be a node which composites data from its left branch onto the rendered page. This may be identified by adding a 'Page' field to the level activation table which identifies nodes which composite their left branch with the rendered page (R is PAGE). This 'Page' field may be ANDed with the Generate L field to provide a single bit per table entry datum which identifies active nodes which are putting data onto the page. The starting node will be the highest priority node among these and can be searched for generating the instructions. This eliminates the overhead of tree traversal for the nodes on the spine which are not providing operations.

Turning now to Fig. 30, there is shown an exemplary expression tree 300 showing those nodes which composite their left branch with the rendered page. As can been seen,

the nodes 302, 304, 306, and 308 composite their left branch onto the rendered branch and the 'Page' field for each of these nodes is set to TRUE(1). This 'page' field may be ANDed with the Generate L field for the same node, which identifies those active nodes putting data onto the page. In the example expression tree, nodes 302, 306 and 308 are active and node 304 inactive. Thus only those sub-trees of active nodes 302, 306 and 308 need be traversed in order to generate the required instructions, thus minimising tree traversal.

Although the preferred embodiment describes the Activity Determination and Instruction Generation Module 500 as an integrated circuit, it may be implemented as an equivalent software module executable on a general purpose processing unit, such as the host processor 2. The software module may form part of a computer program product which may be delivered to a user via a computer readable medium, such as a disk device or computer network.

4.0 Fill Colour Determination Module

The operation of the fill colour determination module 600 will now be described with reference to Fig. 6. Incoming messages 598 from the activity determination and instruction generation module 500, which include set fill data messages, repeat messages, fill priority messages, end of pixel messages, and end of scan line messages, first pass to a fill lookup and control module 604. The fill lookup and control module 604 maintains a current X position counter 614 and a current Y position counter 616 for use by various components of the fill colour determination module 600.

Upon receipt of an end of scan line message, the fill lookup and control module 604 resets the current X counter 614 to zero and increments the current Y counter 616. The end of scan line message is then passed to the pixel compositing module 700.

Upon receipt of a set fill data message, the fill lookup and control module 604 stores the data in the specified location 602 of the fill data table 36.

Upon receipt of a repeat message, the fill lookup and control module 604 increments the current X counter 614 by the count from the repeat message. The repeat message is then passed to the pixel compositing module 700.

Upon receipt of an end of pixel message, the fill lookup and control module 604 again increments the current X counter 614, and the end of pixel message is then passed to the pixel compositing module 700.

Upon receipt of a fill priority message, the fill lookup and control module 604 performs operations which include:

(i) the fill type from the fill priority message is used to select a record size in the table 36;

(ii) the fill table address from the fill priority message, and the record size as determined above, is used to select a record from the fill data table 36;

(iii) the fill type from the fill priority message is used to determine and select a sub-module to perform generation of the fill colour. The sub-modules may include a raster image module 606, a flat colour module 608, a linearly ramped colour module 610, and an opacity tile module 612;

(iv) the determined record is supplied to the selected sub-module 606-612;

(v) the selected sub-module 606-612 uses the supplied data to determine a colour and opacity value;

(vi) the determined colour and opacity is combined with remaining information from the fill colour message, namely the raster operation code, the alpha channel operation code, the source pop flag, and the destination pop flag, to form a colour composite message, which is sent to the pixel compositing module 700 via the connection 698.

In the preferred embodiment the determined colour and opacity is a red, green, blue and opacity quadruple with 8-bit precision in the usual manner giving 32 bits per pixel. However, a cyan, magenta, yellow and black quadruple with an implied opacity, or one of many other known colour representations may alternatively be used. The red, green, blue and opacity case is used in the description below, but the description may also be applied to other cases.

The operation of the raster image module 606, the flat colour module 608, the linearly ramped colour module 610, and the opacity tile module 612 will now be described.

The flat colour module 608 interprets the supplied record as a fixed format record containing three 8-bit colour components (typically interpreted as red, green and blue components) and an 8-bit opacity value (typically interpreted as a measure of the fraction of a pixel which is covered by the specified colour, where 0 means no coverage, that is complete transparency, and 255 means complete coverage, that is, completely opaque). This colour and opacity value is output directly via the connection 698 and forms the determined colour and opacity without further processing.

The linearly ramped colour module 610 interprets the supplied record as a fixed format record containing four sets of constants cx, cy and d, associated with the three

colour and one opacity components, and two position values *xbase* and *ybase* being the coordinates of the reference point of the linear ramp. At the reference point, the colour and opacity components have their associated *d* value.

For each of the four sets, a result value r is computed by combining three constants with the current X and Y coordinates, and the *xbase* and *ybase* constants, using the formula:

$$r = clamp\,(\,cx \times (X - xbase) + cy \times (Y - ybase) + d\,)$$

where the function *clamp* is defined as:

$$clamp\,(x) = \begin{cases} 255 & 255 < x \\ \lfloor x \rfloor & 0 < x < 255 \\ 0 & x < 0 \end{cases}$$

In an alternative implementation, the linearly ramped colour module 610 interprets the supplied record as a fixed format record containing four sets of three constants, *cx*, *cy*, and *d*, being associated with the three colour and one opacity components. For each of these four sets, a result value *r* is computed by combining the three constants with the current X count, *x*, and the current Y count, *y*, using the formula:

$$r = clamp\,(\,cx \times x + cy \times y + d\,)$$

where the function *clamp* is defined as above.

The four results so produced are formed into a colour and opacity value. This colour and opacity value is output directly via the connection 698 and forms the determined colour and opacity without further processing.

Other mathematical calculations giving the same result may be used.

The opacity tile module 612 interprets the supplied record as a fixed format record containing three 8-bit colour components, an 8-bit opacity value, an integer X phase, (*px*), a Y phase, (*py*), an X scale, (*sx*), a Y scale, (*sy*), and a 64 bit mask. These values originate in the display list generation and contained typically in the original page description. A bit address, *a*, in the bit mask, is determined by the formula:

$$a = ((\,x/2^{sx} + px\,) \bmod 8\,) + ((\,y/2^{sy} + py\,) \bmod 8\,) \times 8$$

The bit at the address "*a*" in the bit mask is examined. If the examined bit is one, the colour and opacity from the record is copied directly to the output of the module 612 and forms the determined colour and opacity. If the examined bit is zero, a colour having three zero component values and a zero opacity value is formed and output as the determined colour and opacity.

The raster image module 606 interprets the supplied record as a fixed format record containing six constants, *a*, *b*, *c*, *d*, *xbase* and *ybase*; an integer count of the number of bits (*bpl*) in each raster line of the raster image pixel data 16 to be sampled; and a pixel type. The pixel type indicates whether the pixel data 16 in the raster image pixel data is to be interpreted as one of:

(i) one bit per pixel black and white opaque pixels;

(ii) one bit per pixel opaque black or transparent pixels;

(iii) 8 bits per pixel grey scale opaque pixels;

(iv) 8 bits per pixel black opacity scale pixels;

(v) 24 bits per pixel opaque three colour component pixels; or

(vi) 32 bits per pixel three colour component plus opacity pixels.

Many other formats are possible.

The raster image module 606 uses the pixel type indicator to determine a pixel size (*bpp*) in bits. Then a bit address, *a*, in the raster image pixel data 16 is calculated having the formula:

$$a = bpp * \lfloor a \times (x - xbase) + c \times (y - ybase) \rfloor + bpl \times \lfloor b \times (x - xbase) + d \times (y - ybase) \rfloor .$$

A pixel interpreted according to the pixel type from the record 602 is fetched from the calculated address "*a*" in the raster image pixel data 16. The pixel is expanded as necessary to have three eight bit colour components and an eight bit opacity component. By "expanded", it is meant for example, that a pixel from an eight bit per pixel grey scale opaque raster image would have the sampled eight bit value applied to each of the red, green and blue component, and the opacity component set to fully opaque. This then forms the determined colour and opacity output 698 to the pixel compositing module 700.

As a consequence, the raster pixel data valid within a displayable object is obtained through the determination of a mapping to the pixel image data within the memory 16. This effectively implements an affine transform of the raster pixel data into the object-based image and is more efficient than prior art methods which transfer pixel data from an image source to a frame store where compositing with graphic object may occur.

As a preferred feature to the above, interpolation between pixels in the raster image pixel data 16 may optionally be performed by first calculating intermediate results *p*, and *q* according to the formulae:

$$p = a \times (x - xbase) + c \times (y - ybase)$$
$$q = b \times (x - xbase) + d \times (y - ybase) .$$

Next the bit addresses, a_{00}, a_{01}, a_{10}, and a_{11}, of four pixels in the raster image pixel data 16 are determined according to the formulae:

$$a_{00} = bpp \times \lfloor p \rfloor + bpl \times \lfloor q \rfloor$$

$$a_{01} = a_{00} + bpp$$

$$a_{10} = a_{00} + bpl$$

$$a_{11} = a_{00} + bpl + bpp$$

Next, a result pixel component value, r, is determined for each colour and opacity component according to the formula:

$$r = interp\,(\,interp\,(\,get(a_{00}), get(a_{01}), p),\ interp\,(\,get(a_{10}), get(a_{11}), p),\ q\,)$$

where the function *interp* is defined as:

$$interp\,(a, b, c) = a + (b-a) \times (c - \lfloor c \rfloor)$$

In the above equations, the representation $\lfloor value \rfloor = floor\ (value)$, where a *floor* operation involves discarding the fractional part of *value*.

The *get* function returns the value of the current pixel component sampled from the raster image pixel data 16 at the given bit address. Note that for some components of some image types this can an implied value.

As a preferred feature to the above, image tiling may optionally be performed by using x and y values in the above equations which are derived from the current X and Y counters 614, 616 by a modulus operation with a tile size read from the supplied record.

Many more such fill colour generation sub-modules are possible.

5.0 Pixel Compositing Module

The operation of the pixel compositing module 700 will now be described. Incoming messages from the fill colour determination module 600, which include repeat messages, colour composite messages, end of pixel messages, and end of scan line messages are processed in sequence.

Upon receipt of a repeat message or an end of scan line message, the pixel compositing module 700 forwards the message to a pixel output FIFO 702 without further processing.

Upon receipt of a colour composite message the pixel compositing module 700 typically, and in general terms combines the colour and opacity from the colour composite message with a colour and opacity popped from the pixel compositing stack 38 according to the raster operation and alpha channel operation from the colour composite message. It then pushes the result back onto the pixel compositing stack 38. A

description of the processing performed upon of receipt of a colour composite message is given below.

Upon receipt of an end of pixel message, the pixel compositing module 700 pops a colour and opacity from the pixel compositing stack 38, with the exception that if the stack 38 is empty an opaque white value is used. The resultant colour and opacity is formed into an pixel output message which is forwarded to the pixel output FIFO.

A known compositing approach is that described in "Compositing Digital Images", Porter, T; Duff, T; Computer Graphics, Vol. 18 No. 3 (1984) pp. 253-259. Examples of Porter and Duff compositing operations are shown in Fig. 15. However, such an approach is deficient in that it only permits handling source and destination colour in the intersecting region formed by the composite and, as a consequence, is unable to accommodate the influence of transparency in the intersecting region. This results in the raster operations defined by Porter and Duff as being essentially inoperative in the presence of transparency.

The processing performed by the pixel compositing module 700 upon receipt of a colour composite message will now be described.

Upon receipt of a colour composite message, the pixel compositing module 700 first forms a *source colour and opacity*. This is taken from the colour and opacity provided in the colour composite message unless the pop source flag is set in the colour composite message, in which case the colour is popped from the pixel compositing stack 38 instead. If at this time, or during any pop of the pixel compositing stack, the pixel compositing stack 38 is found to be empty, an opaque white colour value is used without any error indication. Next, a *destination colour and opacity* is popped from the pixel compositing stack 38, except that if the destination pop flag is not set in the colour composite message, the stack pointer is not disturbed during the "pop" operation, in effect making this a read from top of stack 38 instead.

The method of combining the source colour and opacity with the destination colour and opacity will now be described with reference to Figs. 7A to 7C. For the purposes of this description, colour and opacity values are considered to range from 0 to 1, (ie: normalised) although they are typically stored as 8-bit values in the range 0 to 255. For the purposes of compositing together two pixels, each pixel is regarded as being divided into two regions, one region being fully opaque and the other fully transparent, with the opacity value being an indication of the proportion of these two regions. Fig. 7A shows a source pixel 702 which has some three component colour value not shown in the

figure and an opacity value, (*so*). The shaded region of the source pixel 702 represents the fully opaque portion 704 of the pixel 702. Similarly, the non-shaded region in Fig. 7A represents that proportion 706 of the source pixel 702 considered to be fully transparent. Fig. 7B shows a destination pixel 710 with some opacity value, (*do*). The shaded region of the destination pixel 710 represents the fully opaque portion 712 of the pixel 710. Similarly, the pixel 710 has a fully transparent portion 714. The opaque regions of the source pixel 702 and destination pixel 710 are, for the purposes of the combination, considered to be orthogonal to each other. The overlay 716 of these two pixels is shown in Fig. 7C. Three regions of interest exist, which include a source outside destination 718 which has an area of *so* x (1 - *do*), a source intersect destination 720 which has an area of *so* x *do*, and a destination outside source 722 which has an area of (1 - *so*) x *do*. The colour value of each of these three regions is calculated conceptually independently. The source outside destination region 718 takes its colour directly from the source colour. The destination outside source region 722 takes its colour directly from the destination colour. The source intersect destination region 720 takes its colour from a combination of the source and destination colour. The process of combining source and destination colour, as distinct from the other operations discussed above is termed a raster operation and is one of a set of functions as specified by the raster operation code from the pixel composite message. Some of the raster operations included in the preferred embodiment are shown in the Table shown in Fig. 17 of the drawings.

Each function is applied to each pair of corresponding colour components of the source and destination colour to obtain a like component in the resultant colour. Many other functions are possible.

The alpha channel operation from the composite pixel message is also considered. The alpha channel operation is performed using three flags, each of which corresponds to one of the regions of interest in the overlay 716 of the source pixel 702 and the destination pixel 710. For each of the regions, a region opacity value is formed which is zero if the corresponding flag in the alpha channel operation is not set, else it is the area of the region.

The resultant opacity is formed from the sum of the region opacities. Each component of the result colour is then formed by the sum of the products of each pair of region colour and region opacity, divided by the resultant opacity.

The resultant colour and opacity is pushed onto the pixel compositing stack 38.

6.0 Pixel Output Module

The operation of the pixel output module 800 will now be described. Incoming messages are read from the pixel output FIFO, which include pixel output messages, repeat messages, and end of scan line messages are processed in sequence.

Upon receipt of a pixel output message the pixel output module 800 stores the pixel and also forwards the pixel to its output. Upon receipt of a repeat message the last stored pixel is forwarded to the output 898 as many times as specified by the count from the repeat message. Upon receipt of an end of scan line message the pixel output module 800 passes the message to its output.

The output 898 may connect as required to any device that utilizes pixel image data. Such devices include output devices such as video display units or printers, or memory storage devices such as hard disk, semiconductor RAM including line, band or frame stores, or a computer network.

It will be appreciated by those skilled in the art that compositing of objects over larger numbers of levels is also possible by extrapolating the method described herein. Furthermore, it will also be appreciated that the various manipulations shown can be used in different compositing paradigms, including framestore-based systems and other stack-based, line or band-based compositing systems.

It will be apparent from the foregoing that the method and apparatus described provide for the rendering of graphic objects with full functionality demanded by sophisticated graphic description languages without a need for intermediate storage of pixel image data during the rendering process.

Industrial Applicability

It is apparent from the above that the embodiment(s) of the invention are applicable to the computer graphics and printing industries.

The foregoing describes only one embodiment/some embodiments of the present invention, and modifications and/or changes can be made thereto without departing from the scope and spirit of the invention, the embodiment(s) being illustrative and not restrictive.

4. Brief Description of the Drawings

Fig. 1 is a schematic block diagram representation of a computer system incorporating the preferred embodiment;

Fig. 2 is a block diagram showing the functional data flow of the preferred embodiment;

Fig. 3 is a schematic block diagram representation of the pixel sequential rendering apparatus and associated display list and temporary stores of the preferred embodiment;

Fig. 4 is a schematic functional representation of the edge processing module of Fig. 2;

Fig. 5 is a schematic functional representation of the activity determination and instruction generation module of Fig. 2;

Fig. 6 is a schematic functional representation of the fill data determination module of Fig. 2;

Figs. 7A to 7C illustrate pixel combinations between source and destination;

Fig. 8 illustrates a two-object image used as an example for explaining the operation of preferred embodiment;

Figs. 9A and 9B illustrate the vector edges of the objects of Fig. 8;

Fig. 10 illustrates the rendering of a number of scan lines of the image of Fig. 8;

Fig. 11 depicts the arrangement of an edge record for the image of Fig. 8;

Figs. 12A to 12J illustrate the edge update routine implemented by the arrangement of Fig. 4 for the example of Fig. 10;

Figs. 13A to 13E illustrate how large changes in X coordinates contribute to spill conditions and how they are handled;

Figs. 14A to 14D provide a comparison between two prior art edge description formats and that used in the preferred embodiment;

Fig. 15 depicts the result of a number of compositing operations;

Fig. 16 shows an exemplary table of edge records for storage in the edge record store of Fig. 2;

Fig. 17 shows a table of some of the raster operations for use in the preferred embodiment;

Figs. 18A and 18B show a simple compsoting expression illustrated as an expression tree and a corresponding description;

Fig. 19 shows an exemplaty expression tree for implementing a series of Porter and Duff compositing operations on objects A,B,C and D and a corresponding instruction list;

Fig. 20 shows the expression tree of Fig. 19 where object C is active and a corresponding instruction list;

Fig. 21 shows a level activation table generated by the level activation table generator of Fig. 5 for the expression tree of Fig. 20;

Fig. 22 shows the logic circuit used in the level activation table update module of Fig. 5;

Fig. 23 shows an updated level activation table for the expression tree of Fig. 20 where object C is active;

Fig. 24 shows a further embodiment of the logic circuit used in the level activation table update module of Fig. 5;

Fig. 25 shows a level activation table for use with the further embodiment of Fig. 24;

Fig. 26 shows a truth table for the IN compositing operation;

Fig. 27 shows a truth table for the CLIP IN compositing operation;

Fig. 28 shows a truth table for the OUT compositing operation;

Fig. 29 shows a truth table for the CLIP OUT compositing operation; and

Fig. 30 illustrates an exemplary expression tree showing nodes which composite their left branch with the rendered page.

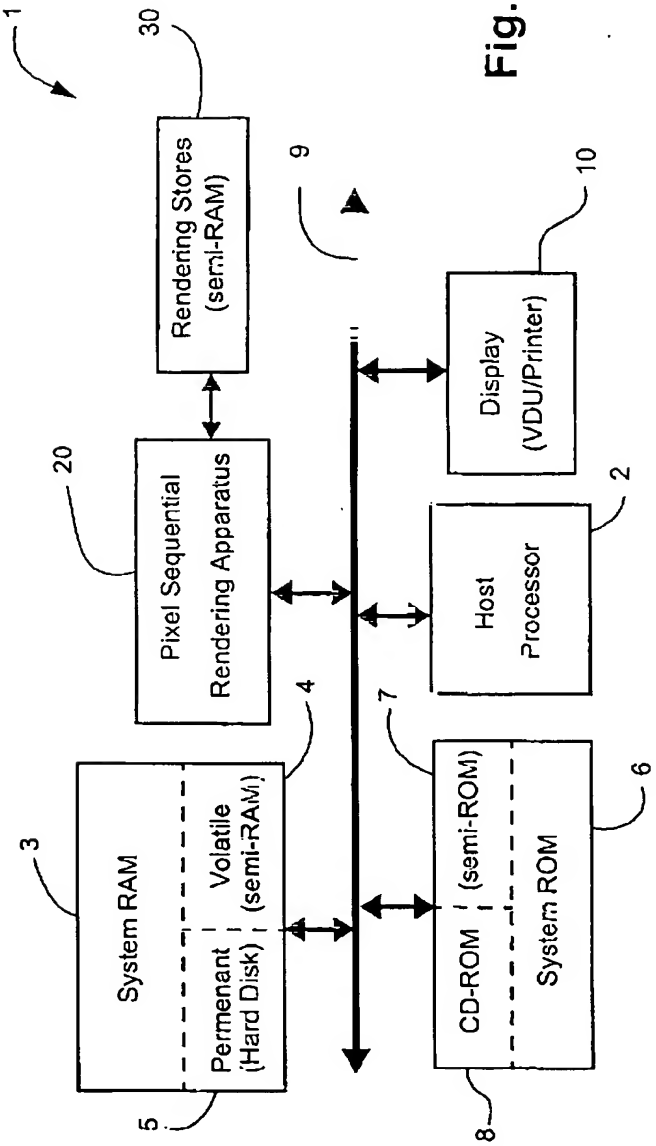

1
30
Rendering Stores (semi-RAM)
9
10
Display (VDU/Printer)
20
Pixel Sequential Rendering Apparatus
2
Host Processor
4
7
3
System RAM
Volatile (semi-RAM)
Permenant (Hard Disk)
CD-ROM (semi-ROM)
System ROM
6
5
8

Fig. 1

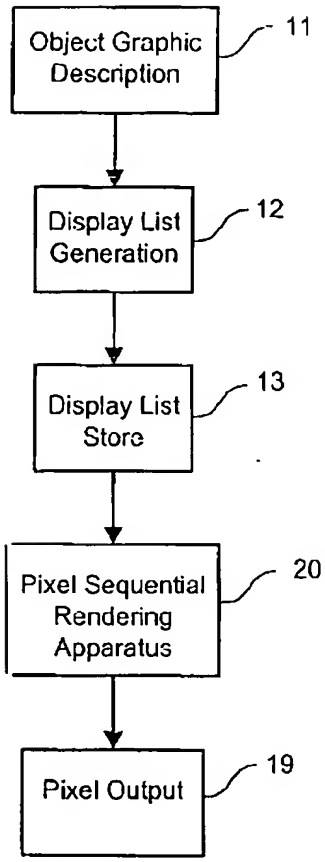

Object Graphic Description
11
Display List Generation
12
Display List Store
13
Pixel Sequential Rendering Apparatus
20
Pixel Output
19

Fig. 2

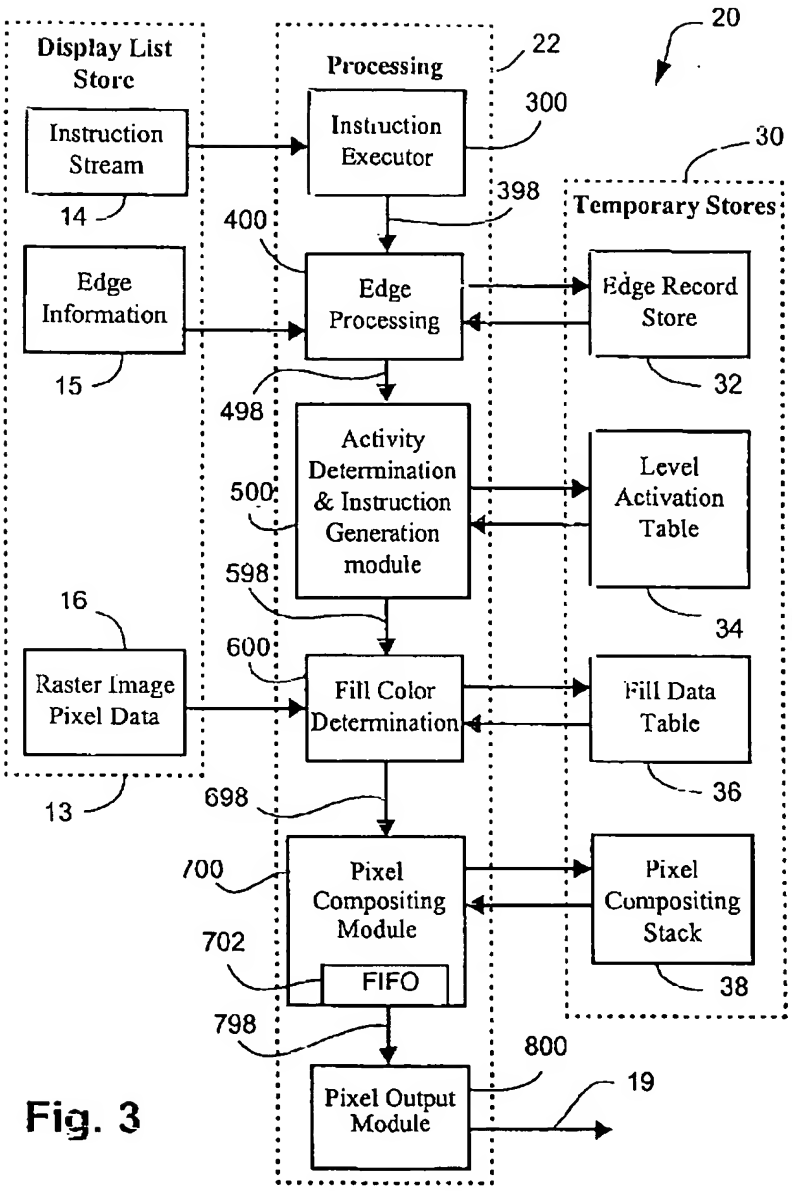

Display List Store
Instruction Stream
14
Edge Information
15
16
Raster Image Pixel Data
13
Processing
22
20
Instruction Executor
300
30
Temporary Stores
398
400
Edge Processing
Edge Record Store
32
498
Activity Determination & Instruction Generation module
500
Level Activation Table
34
598
600
Fill Color Determination
Fill Data Table
36
698
700
Pixel Compositing Module
Pixel Compositing Stack
38
702
FIFO
798
800
Pixel Output Module
19

Fig. 3

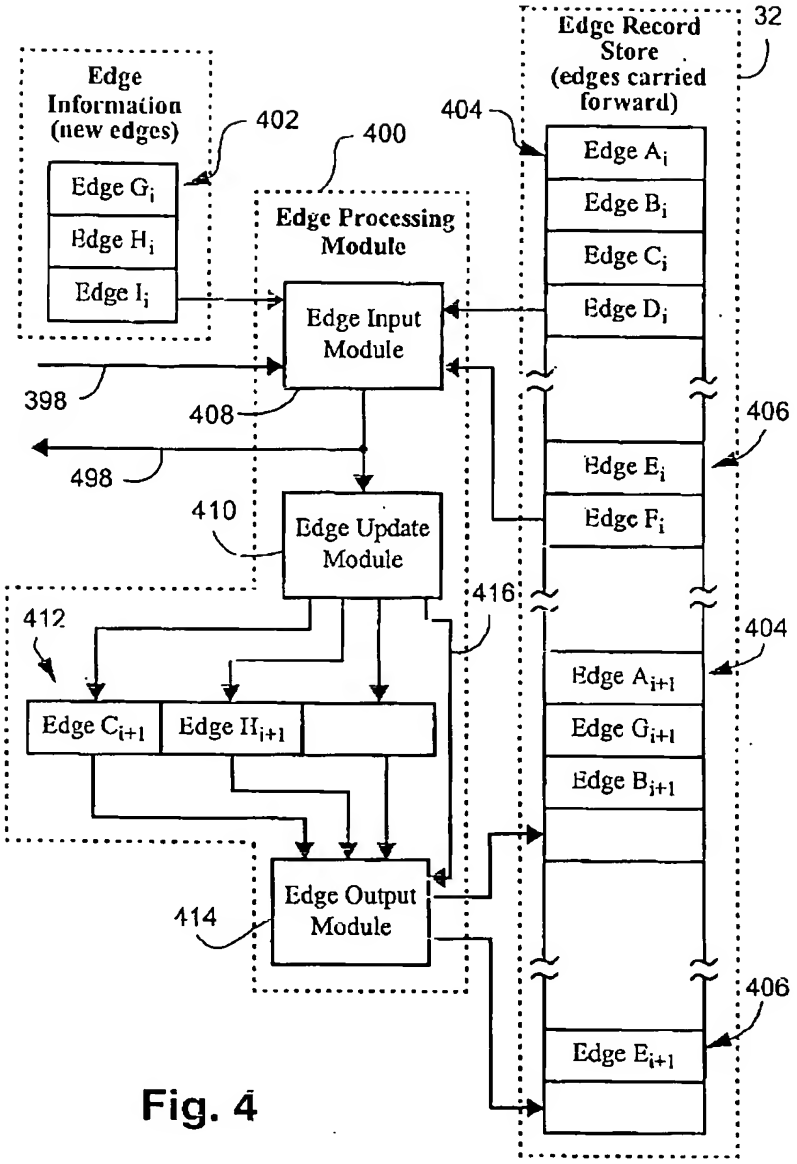

Edge Record Store (edges carried forward)
32
Edge Information (new edges)
402
400
404
Edge G_i
Edge H_i
Edge I_i
Edge Processing Module
Edge Input Module
Edge A_i
Edge B_i
Edge C_i
Edge D_i
398
408
406
498
Edge E_i
Edge F_i
410
Edge Update Module
416
412
404
Edge A_{i+1}
Edge C_{i+1}
Edge H_{i+1}
Edge G_{i+1}
Edge B_{i+1}
Edge Output Module
414
406
Edge E_{i+1}

Fig. 4

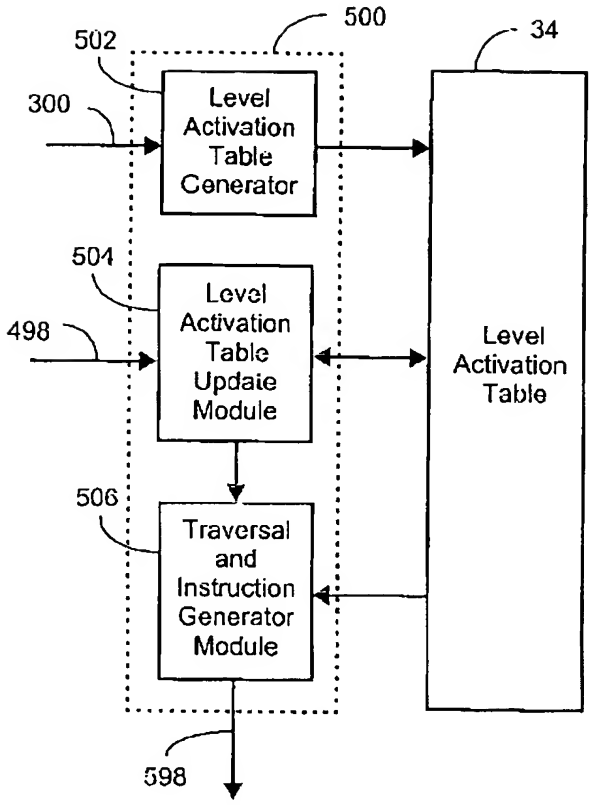

500
502
34
300
Level Activation Table Generator
504
498
Level Activation Table Update Module
Level Activation Table
506
Traversal and Instruction Generator Module
598

Fig. 5

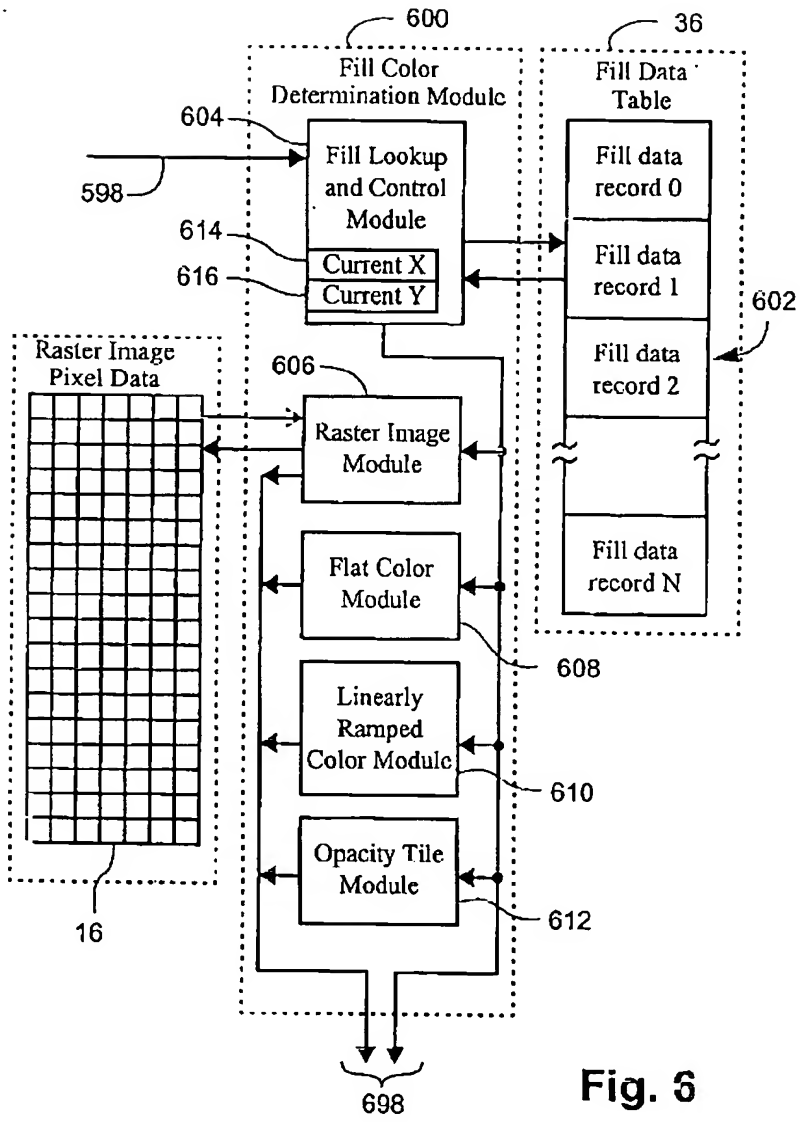

600
36
Fill Color Determination Module
Fill Data Table
604
Fill Lookup and Control Module
598
614
Current X
616
Current Y
Fill data record 0
Fill data record 1
Fill data record 2
Fill data record N
602
Raster Image Pixel Data
606
Raster Image Module
Flat Color Module
608
Linearly Ramped Color Module
610
Opacity Tile Module
612
16
698

Fig. 6

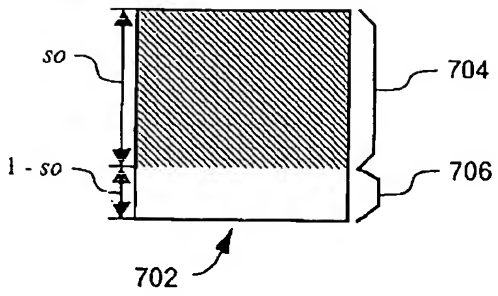


Fig. 7A

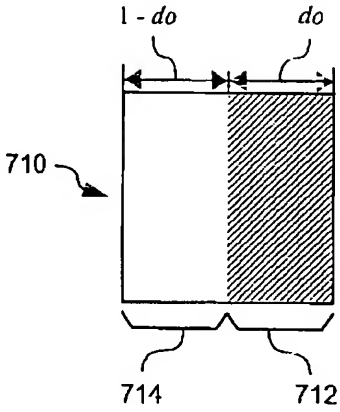


Fig. 7B

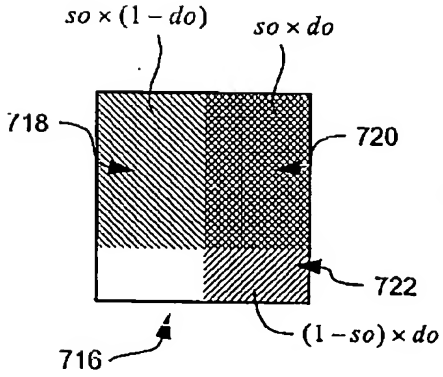


Fig. 7C

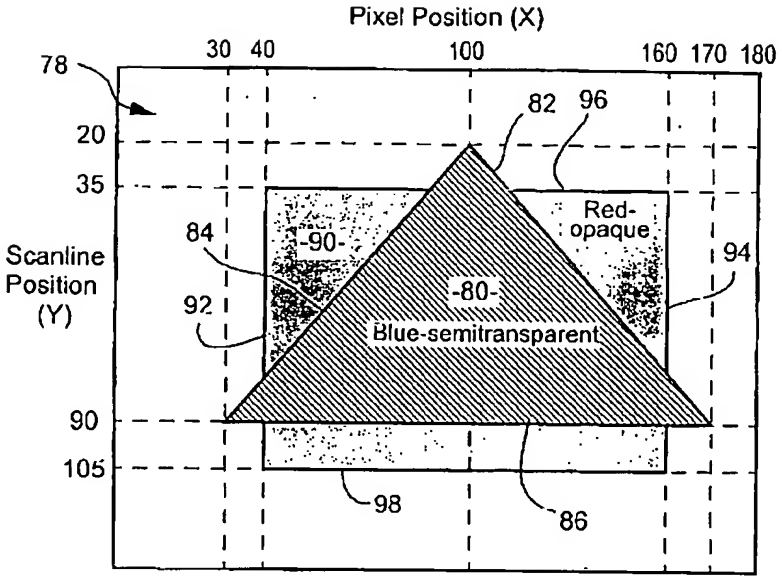


Fig. 8

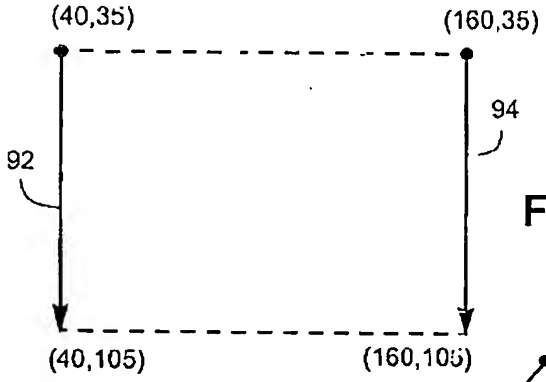


Fig. 9A

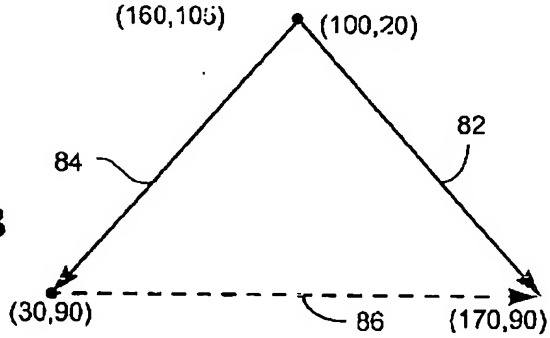


Fig. 9B

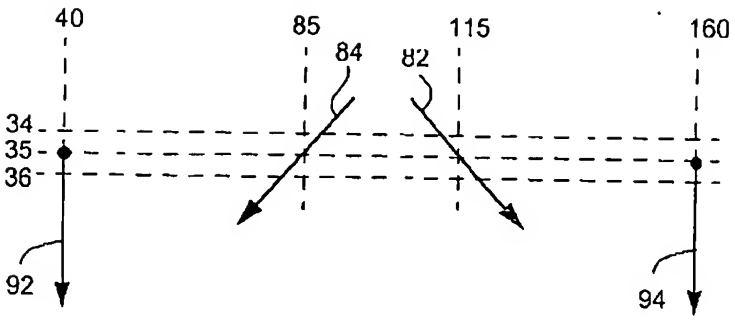

40
85
84
82
115
160
34
35
36
92
94

Fig. 10

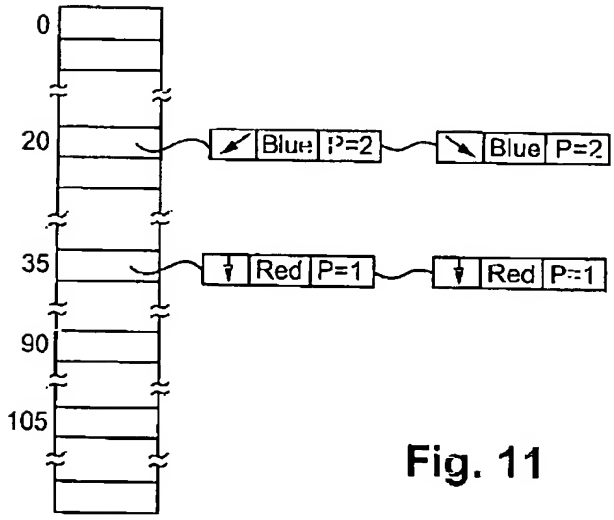

0
20
Blue P=2
Blue P=2
35
Red P=1
Red P=1
90
105

Fig. 11

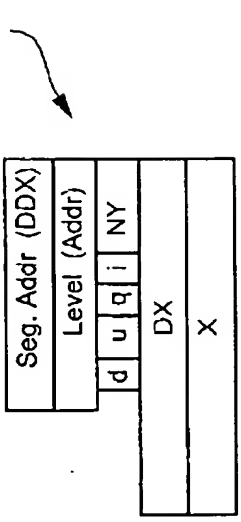


Fig. 12A

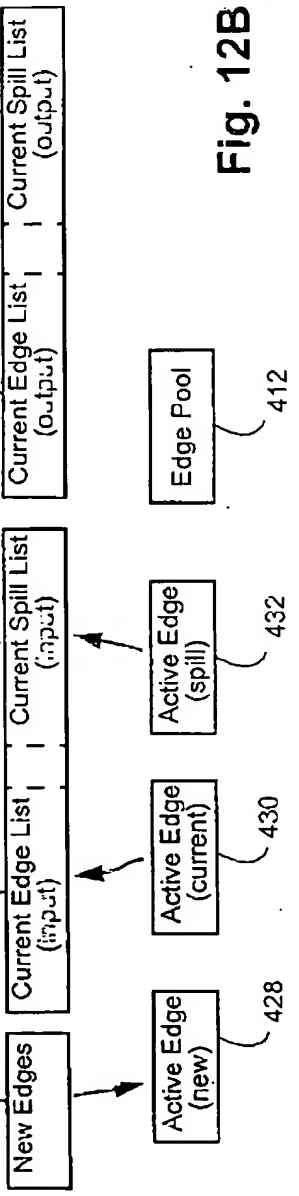


Fig. 12B

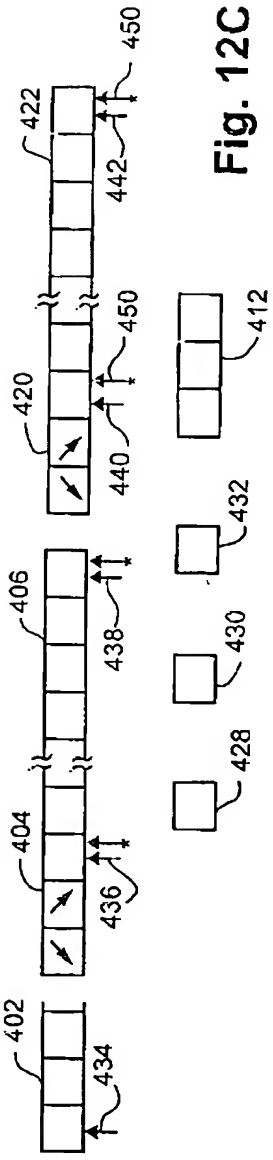


Fig. 12C

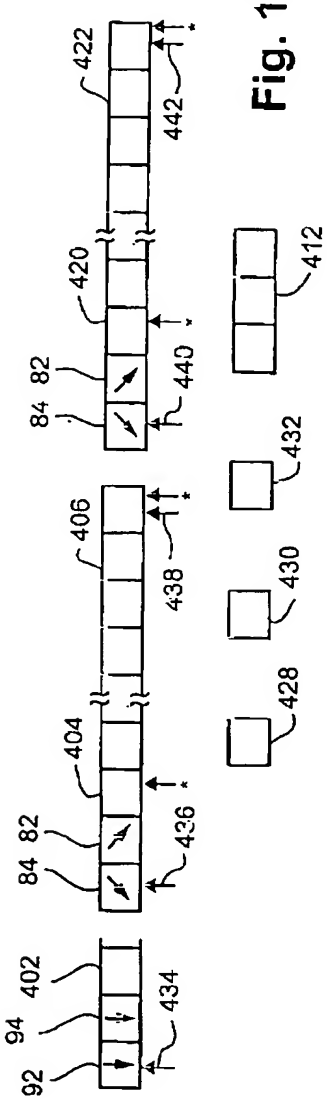


Fig. 12D

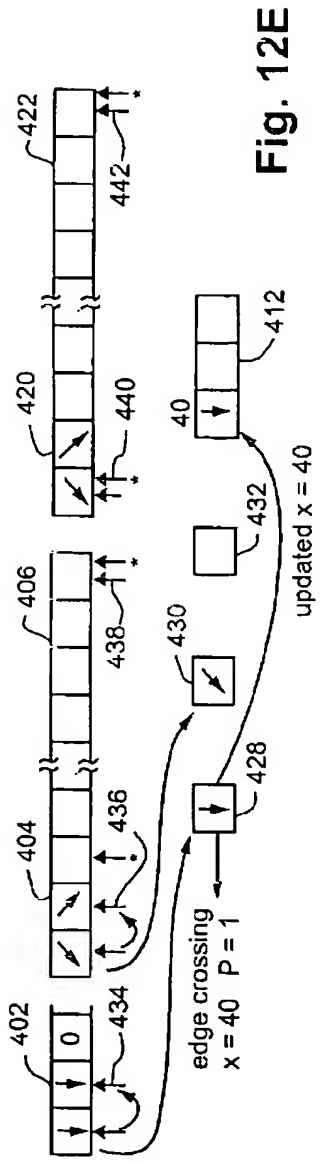


Fig. 12E

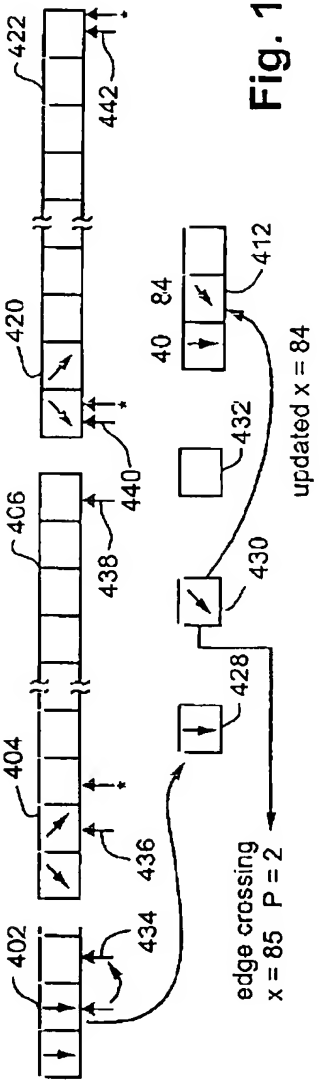


Fig. 12F

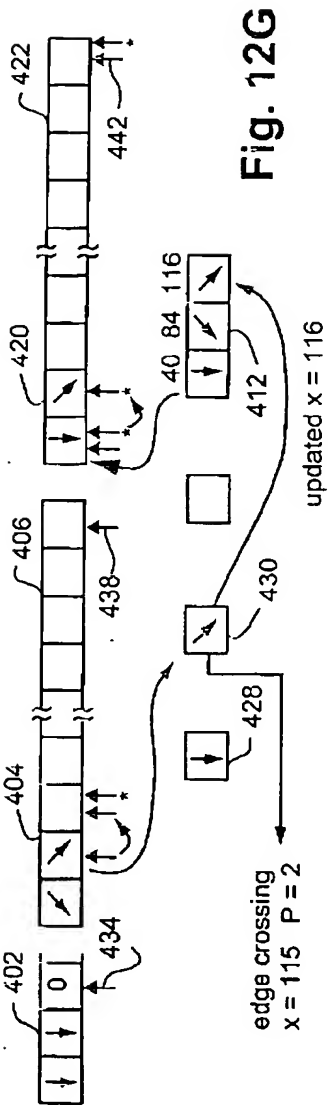


Fig. 12G

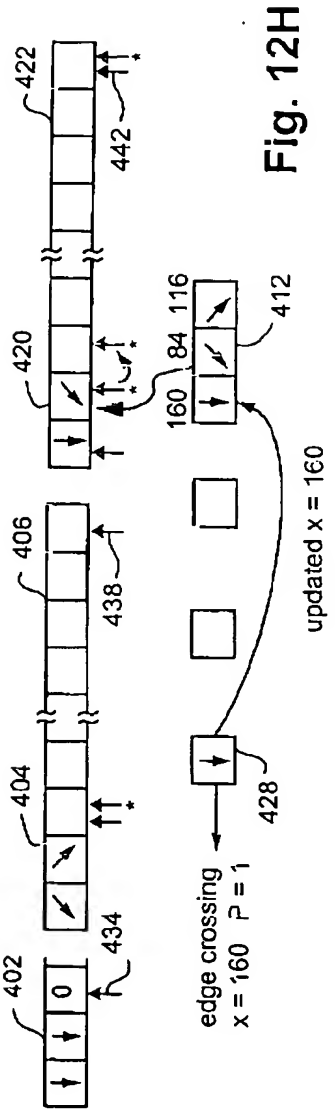


Fig. 12H

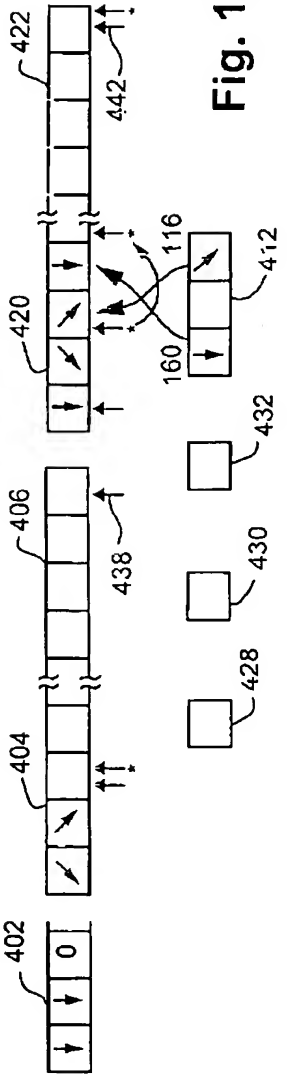


Fig. 12I

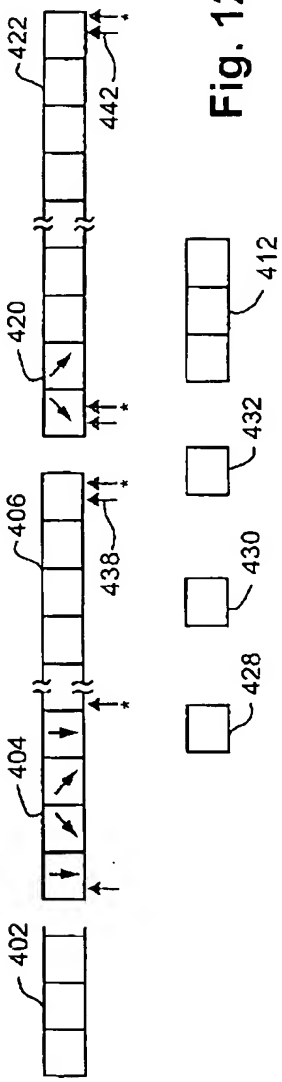


Fig. 12J

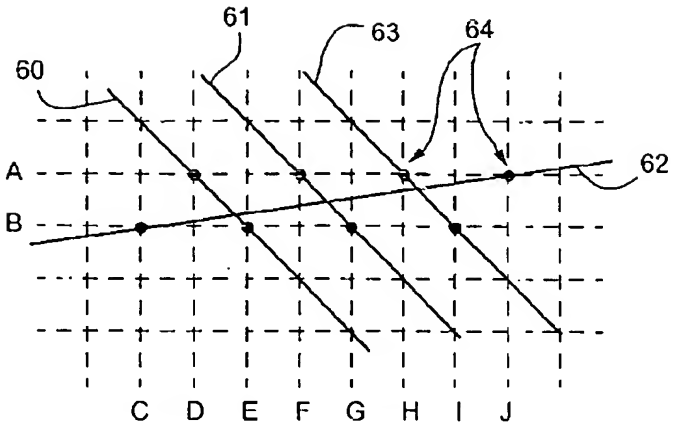


Fig. 13A

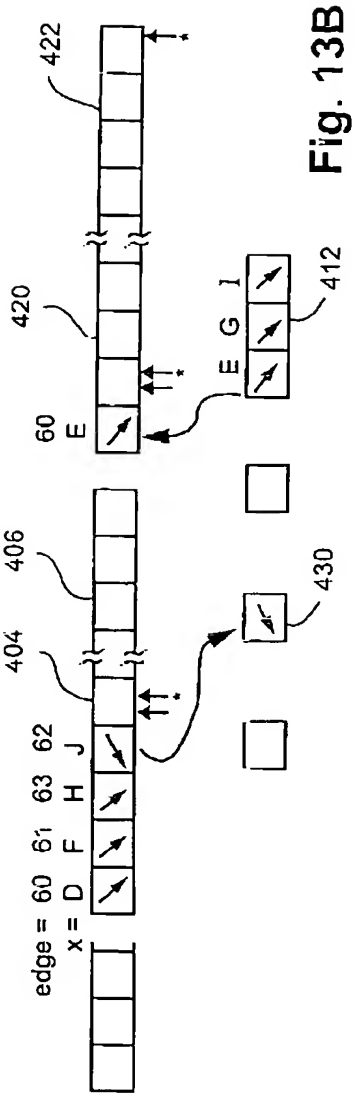


Fig. 13B

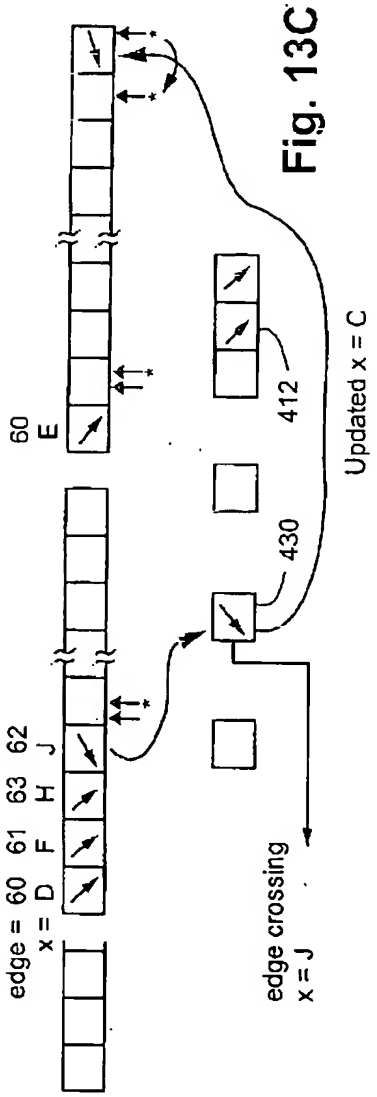


Fig. 13C

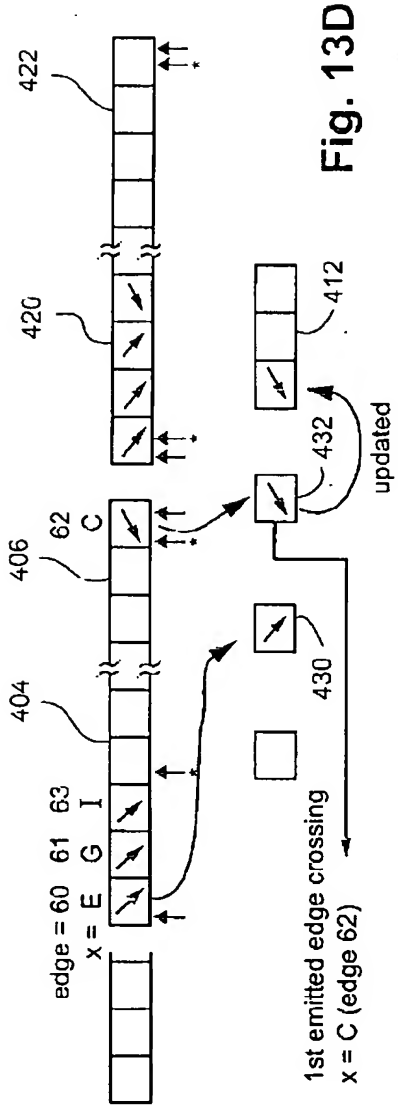


Fig. 13D

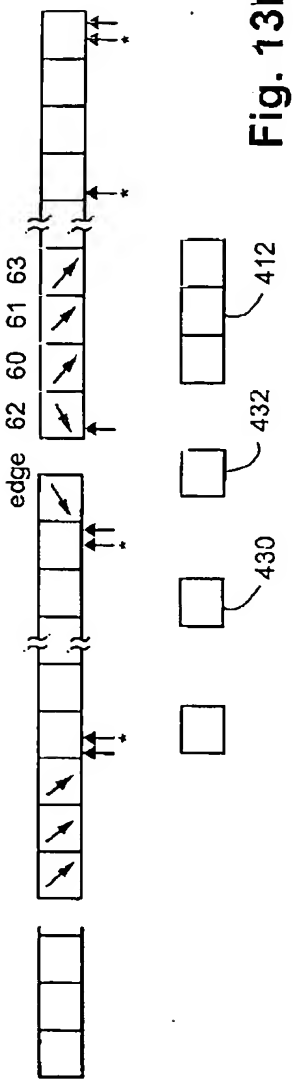


Fig. 13E

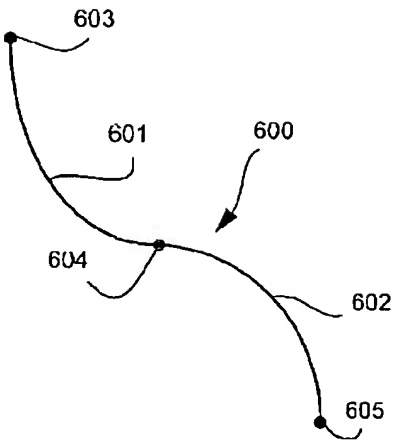


Fig. 14A
(Prior Art)

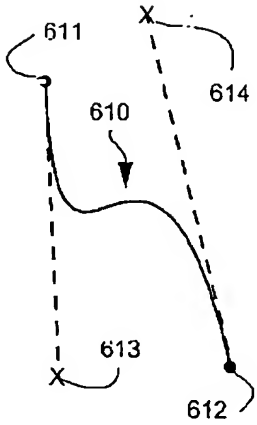


Fig. 14B
(Prior Art)

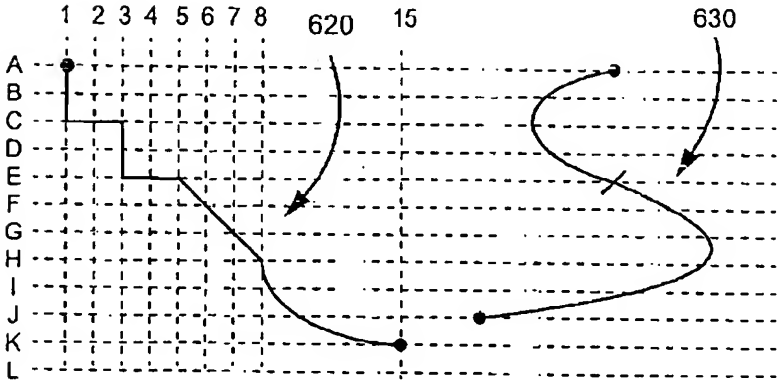


Fig. 14C

Fig. 14D

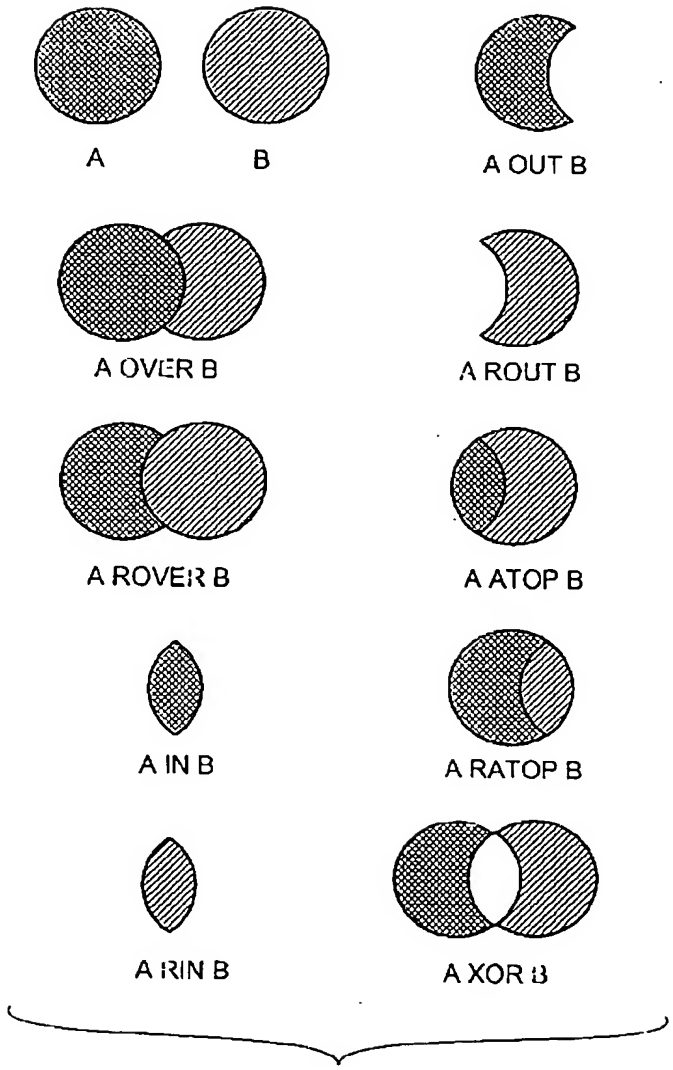

A
B
A OUT B
A OVER B
A ROUT B
A ROVER B
A ATOP B
A IN B
A RATOP B
A RIN B
A XOR B

Fig. 15

Edge 84	Edge 92
X = 100	X = 40
NY = 70	NY = 70
DX = 1	DX = 0
DDX = 0	DDX = 0
P = 1	P = 0
DIR = (-)	DIR = (+)
ADD = (irrelevant in this example)	ADD = (irrelevant in this example)

Fig. 16

Raster operation code	Operation -	Comment
0x00	r = 0	BLACKNESS
0x01	r = src & dest	SRCAND
0x02	r = src & ~dest	SRCERASE
0x03	r = src	SRCCOPY
0x04	r = ~src & dest	
0x05	r = dest	NOP
0x06	r = src ^ dest	SRCINVERT
0x07	r = src | dest	SRCPAINT
0x08	r = ~(src | dest)	NOTSRCERASE
0x09	r = ~(src ^ dest)	
0x0a	r = ~dest	DSTINVERT
0x0b	r = src | ~dest	
0x0c	r = ~src	NOTSRCCOPY
0x0d	r = ~src | dest	MERGEPAINT
0x0e	r = ~(src & dest)	
0x0f	r = 0xff	WHITENESS
0x10	r = min(src, dest)	
0x11	r = max(src, dest)	
0x12	r = clamp(src + dest)	
0x13	r = src	
0x14	r = clamp(src - dest)	
0x15	r = dest	
0x16	r = clamp(dest - src)	
0x17	r = clamp(src + dest) where dest is signed	
0x18	r = threshold (dest, src)	
0x19	r = threshold (src, dest)	
0x1a	r = ~ dest	
0x1b	o = luminance (dest, src)	
0x1c	r = ~src	
0x1d	o = ckey(dest; src +/- o)	

Fig. 17

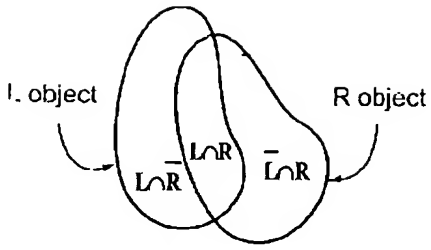


Fig. 18A

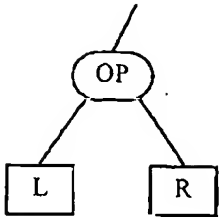


Fig. 18B

Fig. 19

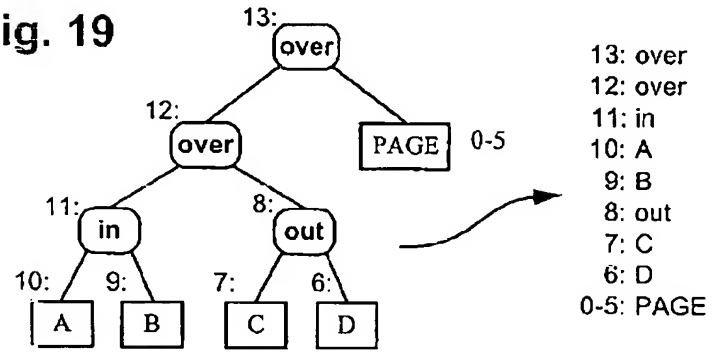
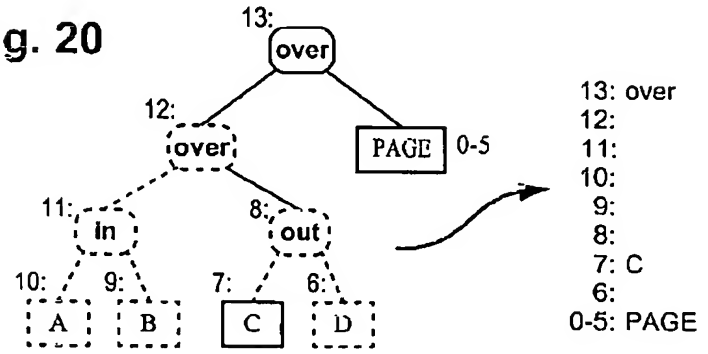


Fig. 20



Index	L Active	R Active	L∩$\bar{R}$ reqd	$\bar{L}$∩R reqd	Leaf/ Operator Entry	Node Active	Parent	Node is L	Generate L	Generate R	L∩R op used	R Branch Index
13	0	1	1	1	over	1	?	0	0	1	0	5
12	0	0	1	1	over	0	13	1	0	0	0	8
11	0	0	0	0	in	0	12	1	0	0	0	9
10					leaf A	0	11	1				
9					leaf B	0	11	0				
8	0	0	1	0	out	0	12	0	0	0	0	6
7					leaf C	0	8	1				
6					leaf D	0	8	0				
PAGE												

Fig. 21

(126)|2002-56396 (P2002-56396A)

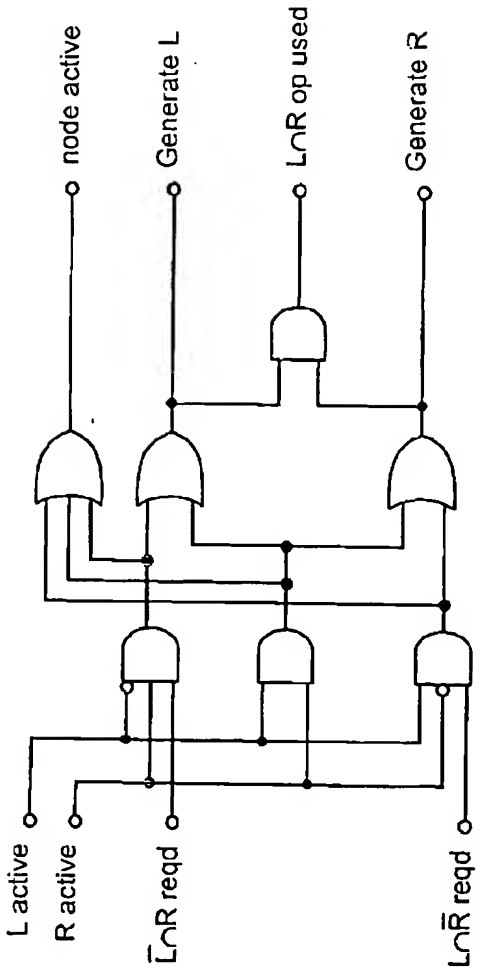

node active
Generate L
L∩R op used
Generate R
L active
R active
L∩R reqd
L∩R̄ reqd

Fig. 22

Index	L Active	R Active	L∩$\bar{R}$ reqd	$\bar{L}$∩R reqd	Leaf/ Operator Entry	Node Active	Parent	Node is L	Generate L	Generate R	L∩R op used	R Branch Index
13	[illegible]	1	1	1	over	1	?	0	1	1	1	5
12	0	[illegible]	1	1	over	1	13	1	0	1	0	8
11	0	0	0	0	in	0	12	1	0	0	0	9
10					leaf A	0	11	1				
9					leaf B	0	11	0				
8	[illegible]	0	1	0	out	1	12	0	1	0	0	6
7					leaf C	1	8	1				
6					leaf R	0	8	0				
PAGE												

Fig. 23

(128)I2002-56396(P2002-56396A)

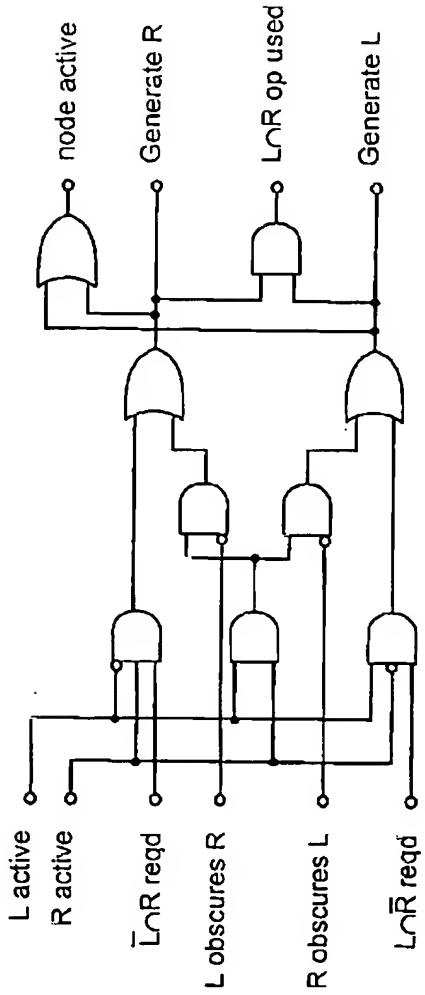

node active
Generate R
L∩R op used
Generate L
L active
R active
L∩R reqd
L obscures R
R obscures L
L∩R̄ reqd

Fig. 24

Index	L Active	R Active	L∩R reqd	L∩R reqd	L obscures R	R obscures L	Leaf/ Operator Entry	Node Active	Parent	Node is L	Gene-rate L	Gene-rate R	L∩R op used	R Branch Index
			0	0	0	0	in							
			0	0	1	0	(CLIP IN)							
			1	0	0	0	out							
			1	0	1	1	(CLIP OUT							

Fig. 25

(130)I2002-56396 (P2002-56396A)

L	T	T	T	L	L	L	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	0	1	1	0	1	1
Generate R	0	0	0	0	0	0	0	0	0
Result	T	T	T	T	L	L	T	O	O

Fig. 26

L	T	T	T	L	L	L	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	0	1	1	0	1	1
Generate R	0	0	0	0	1	1	0	1	1
Result	T	T	T	T	LR	L	T	R	O

Fig. 27

L	T	T	T	L	L	L	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	1	1	1	1	1	1
Generate R	0	0	0	0	1	1	0	1	1
Result	T	T	T	L	L(1-R)	T	O	(1-R)	T

Fig. 28

L	T	T	T	L	L	L	O	O	O
R	T	R	O	T	R	O	T	R	O
Generate L	0	0	0	1	0	0	1	0	0
Generate R	0	0	0	0	0	0	0	0	0
Result	T	T	T	L	T	T	O	T	T

Fig. 29

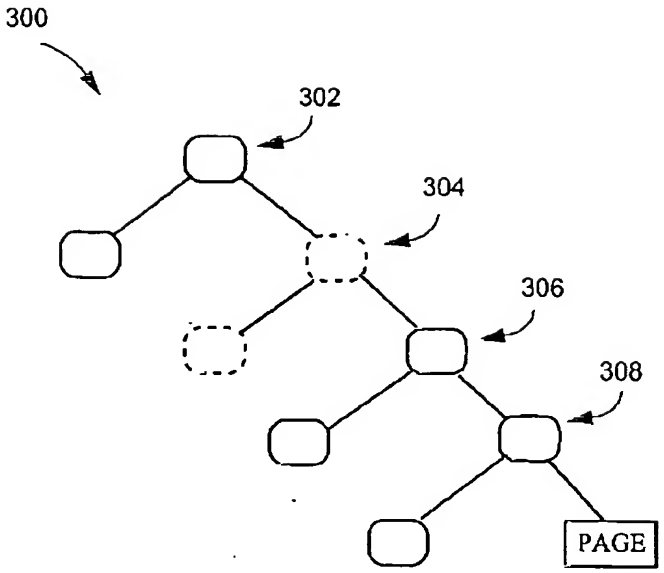


Fig. 30

1. Abstract

Disclosed are methods, apparatus (1) and computer readable medium for generating instructions for a directed adjacency graph, such as an expression tree, into a raster pixel image having a plurality of scan lines and a plurality of pixel locations on each scan line. The expression tree comprises one or more parent nodes and one or more leaf nodes. The parent nodes each representing an operator and each having branches to respective descendent nodes. The leaf nodes each representing a graphic object. The apparatus comprises means for scanning a plurality of pixel locations (300). The apparatus further comprising a module (504) for determining, for each of the scanned pixel locations, a portion of the expression tree in accordance with the activity of the operators, where the portion of the expression tree is that portion which passes data up the tree. The apparatus also comprises a Module (506) for generating instructions for the determined portion of the expression tree, wherein operator instructions are generated for those operators of the determined portion of the expression tree having active branches and wherein leaf instructions are generated for those graphic objects which are active at the scanned pixel location.

2. Representative Drawing

Fig. 5